5 DITTRICH

MICRO APPLICATION

ATARI ST

PEEKS ET POKES DE L'ATARI ST

UN LIVRE DATA BECKER

5 DITTRICH

MICRO APPLICATION

ATARI ST

PEEKS ET POKES DE L'ATARI ST

UN LIVRE DATA BECKER

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
9, Rue Jacob
75006 PARIS

(c) Reproduction interdite sans l'autorisation de
MICRO APPLICATION

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-052-5

(c) 1985 DATA BECKER
Merowingerstrasse, 30
4000 DUSSELDORF
R.F.A.

Traduction Française assurée par Mr BRONNENKANT

(c) 1985 MICRO APPLICATION
13 Rue Sainte Cécile
75009 PARIS

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

TABLE DES MATIERES

PREFACE 1

1. APERCU DE L'INTERIEUR 3

1.1 Configuration interne 3

1.2 Les interfaces 6
1.2.1 Les interfaces parallèles 7
1.2.2 Les interfaces séries 9
1.2.3 Le port des unités de disquettes 10
1.2.4 L'interface MIDI 12
1.2.5 Le port d'extension de la ROM 13
1.2.6 Le port souris/joystick 14
1.2.7 Le port moniteur 14

1.3 Un clavier intelligent 19
1.3.1 L'ensemble des commandes 20
1.3.2 L'interrogation de la position du joystick 21
1.3.3 Le pilotage du curseur avec la souris 21
1.3.4 La gestion de la date et de l'heure 22
1.3.5 La scrutation du clavier 23

1.4 La souris 24
1.4.1 La souris comme crayon à dessin 25

2. LES MEMOIRES DE DONNEES 27

2.1 La mémoire interne 27
2.1.1 Configuration de la mémoire principale de l'ATARI ST 29
2.1.2 Adresses des I/O 30
2.1.3 Les pointeurs erreurs 31
2.1.4 Les pointeurs 32
2.1.5 Les piles ou stacks 36

2.2 Les disquettes 37
2.2.1 Accès direct aux fichiers sur disquette 38
2.2.2 Sauvegarde d'une image graphique 39

3. LA REPRESENTATION INTERNE DES NOMBRES ET LEUR MANIPULATION 40

3.1 Conversion des systèmes numériques 41
3.2 Manipulation de bits (pour la gestion du joystick par ex.) 42
3.3 Les opérations logiques 43

4. LE SYSTEME D'EXPLOITATION 47

4.1 Le Tramiel-Operator-System 47
4.1.1 Le BIOS 48
4.1.2 Les variables systèmes 49
4.1.3 L'écran TOS 51

4.2 Le GEM 53
4.2.1 La programmation du GEM en BASIC 54
4.2.2 Gestion de la souris 56
4.2.3 Modification de la forme de la souris 57
4.2.4 Changement du type de caractère 61
4.2.5 Texte graphique 63

4.3 Interprèteur/Compilateur 66

5. LA PREPARATION DU BUREAU 69

5.1 Configuration des interfaces 69

6. LES LANGAGES DE PROGRAMMATION 72

6.1 Dr. LOGO 72
6.2 BASIC ATARI 74
6.3 'C' 76

6.4 Le langage machine du MC 68000 80
6.4.1 Combinaison avec le BASIC 93

7. PROGRAMMATION DE L'ATARI EN BASIC 101

7.1 Le graphisme 103
7.1.1 Cercles, ellipses et rectangles 104
7.1.2 Réalisation de texte graphique 108
7.1.3 Lignes droites et marques 112
7.1.4 Remplissage de surfaces quelconques 115
7.1.5 Redéfinition du motif de remplissage 116
7.1.6 Affichage de marques à l'écran 117
7.1.7 Test de la couleur d'un point à l'écran 119
7.1.8 Mélange de couleurs 120

7.2 Bruit et musique 121
7.3 Programmation des fenêtres et des menus 128

7.4 Traitement de texte 136
7.4.1 Masque de saisie 138

7.5 Pilotage : souris/joystick 139

7.6 Entrées/sorties 141
7.6.1 Pilotage de l'imprimante 141
7.6.2 Utilisation des disquettes 143
7.6.3 Communication 149

7.7 Redéfinition de caractères 152

ANNEXES :

1. Glossaire 159
2. Peek et Poke Utiles 169
3. Table des Mots Clés 170

PREFACE

Avec l'ATARI ST, le marché de l'ordinateur personnel s'enrichit d'un système qui offre des possibilités nouvelles. L'étendue de ses capacités, son confort d'utilisation et, avant tout, son rapport qualité/prix en font l'un des meilleurs de sa catégorie. Il a d'autres atouts : un clavier bien disposé, de nombreuses interfaces, une capacité mémoire de 512 Koctets (et même plus) , et un système d'exploitation GEMDOS dont l'apprentissage et l'utilisation sont, grâce à la souris, un jeu d'enfant.

Mais peut-on mettre à profit cet ordinateur pour réaliser ses propres idées ? Certes, le manuel d'utilisation qui est fournit avec l'appareil décrit l'emploi (de la plupart) des fonctions du système, mais beaucoup de questions sont laissées en suspens.

Le présent ouvrage répond à ces questions. Il donne un aperçu de la 'vie intérieure' de votre ordinateur. Il représente, pour ainsi dire, un guide de voyage à travers le 'monde fantastique de l'ATARI ST'. Grâce à de nombreux exemples, les capacités de l'ordinateur vous sont entièrement dévoilées. Il vous est ainsi donné la possibilité de créer vos propres applications de l'ATARI ST.

Afin de pouvoir prendre de telles initiatives, certaines instructions spéciales sont indispensables. Je fais ici référence au titre de cet ouvrage. PEEK et POKE sont des instructions du langage de programmation BASIC, qui permettent d'accéder directement aux fonctions internes de l'ordinateur. Vous pourrez ensuite, à l'aide de tables contenant les paramètres importants du système, entreprendre de manipuler le puissant système d'exploitation de l'ATARI ST.

De plus, les possibilités de communication avec l'extérieur sont également examinées. L'échange de données avec d'autres ordinateurs, la connexion et l'emploi d'un modem téléphonique seront étudiés ainsi que le pilotage de périphériques externes. La connexion d'un lecteur de disquettes supplémentaire ou d'une imprimante sera également traitée.

Avant donc que vous ne testiez les exemples d'applications de ce livre, je voudrais encore attirer votre attention sur un point : l'ordinateur dont il est question dans cet ouvrage est le premier de sa série. Par conséquent, son système d'exploitation ne réside pas encore en mémoire centrale, mais est chargé à partir de la disquette. C'est pourquoi, si vous possédez une version plus récente de cet ordinateur, certaines adresses indiquées ici peuvent se trouver modifiées. Il en est de même pour la version de BASIC dans laquelle ont été développé les exemples de programmes. Les interpréteurs BASIC définitifs seront sans doute plus courts. Leur implantation dans les versions futures auront naturellement lieu lorsque ATARI mettra sur le marché de plus grandes inovations. Cependant, dans toute la série ST, rien ne fait obstacle à l'application de ce livre.

Mais abrégeons cette introduction et entrons dans le vif du sujet ! Il me reste encore à vous souhaiter beaucoup de succès dans la réalisation de vos projets, à l'aide de cet ouvrage.

Pour terminer, j'aimerais encore remercier mes camarades d'étude et amis pour leur collaboration à la réalisation de ce livre. C'est grâce à eux qu'ont été recueillies une grande quantité des informations et des tuyaux qui vous sont donnés ici.

Novembre 1985

Stephan Dittrich

1. APERCU DE L'INTERIEUR

Un nouvel ordinateur exerce toujours un immense attrait. De ce fait, on est tout naturellement tenté de s'asseoir devant la console et de commencer à programmer. Malheureusement, il arrive fréquemment qu'on néglige les spécificités de l' ordinateur en question et qu'on le considère exactement comme les autres.

Il serait vraiment regrettable d'adopter cette attitude devant l'ATARI ST. En effet, quasiment aucun autre ordinateur personnel n'offre de telles possibilités, au point d'étonner à chaque fois même les programmeurs les plus expérimentés. La puissance du GEM, dont la facilité d'utilisation rend cet ordinateur vraiment sympathique, n'est pas la seule de ces spécificités. La conception technique de cet ordinateur elle-même lui permet d'être employé dans n'importe quel but. Pour s'en rendre compte, il suffit de jeter un coup d'oeil à l'intérieur de l'ATARI ST et d'examiner les éléments qui sont à la base de l'ordinateur, à savoir les composants eux-mêmes. Risquons nous donc à l'intérieur de l'ordinateur.

1.1 CONFIGURATION INTERNE

Le premier coup d'oeil sur le nouvel ATARI ST ne permet pas de deviner ce que cache son bati. On entend souvent dire : "Cet ordinateur a vraiment un bel aspect extérieur, mais où se trouve toute l'éléctronique ? Il ne doit pas contenir grand chose !".

Erreur ! Dans le boitier relativement petit du ST (comparé par ex. à celui de l' IBM PC) se cache énormément d'electronique . Si l'on regarde à l'intérieur, on est surpris qu'un ordinateur puisse être si performant avec des circuits aussi peu encombrants. ATARI a en effet fait appel aux techniques les plus récentes. Voyons donc ce dont il est capable, et comment ou avec quoi il le réalise.

Il y a tout d'abord le microprocesseur lui-même. Il représente le cerveau de l'ordinateur. Il s'agit ici du MC 68000 de Motorola, c'est le plus grand composant sur la plaque. Ce microprocesseur possède de remarquables indices de performance :

* 17 registres de données et d'adresses de 32 bits.
* une zone adressable de 16 Mégaoctets.
* 56 instructions machines
* 14 modes d'adressage

Il travaille avec des mots de 32 bits, qui peuvent donc atteindre la valeur : 2^32 = 4.294.967.296. Les bus de données ont 16 bits, c'est pourquoi on le désigne aussi comme un microprocesseur 16 bits. Le fait de manipuler des grands nombres offre un très grand avantage, à savoir une plus grande vitesse d'exécution. Le MC 68000 peut ainsi traiter beaucoup d'informations (du texte, par ex.) en un minimum de temps. En outre, la fréquence de l'horloge est de 8 Mégahertz. Le microprocesseur exécute donc 8 millions d'instructions machines simples à la seconde. Ce chiffre reste cependant théorique, car beaucoup d'instructions machines nécessitent plusieurs tops horloges. Il n'en reste pas moins que la vitesse d'exécution est énorme. On peut s'en rendre compte, quand, par exemple, l'ATARI ST affiche une image graphique à l'écran.

Le microprocesseur est donc le cerveau de l'ordinateur. Les autres composants sont des exécutants qui assistent le 'boss'. Il existe un deuxième composant 'intelligent' dans le ST, dans le clavier plus précisément. C'est le HD 6301 V1. Ses fonctions sont très importantes. Les voici : scrutation et gestion du clavier y compris des touches-fonctions F1 à F10, gestion de la souris et de ses mouvements, ainsi que la gestion des joysticks. De plus, le 6301 possède une horloge 'véritable' qui compte en seconde. La multiplicité des possibilités offertes par ce clavier à l'utilisateur sont telles qu'un chapitre entier lui est consacré, voir paragraphe 1.3, 'un clavier intelligent'.

Revenons cependant au calculateur principal. Il existe encore ici quelques composants qui sont nécessaires au ST. Les circuits intégrés utilisés ici (désignés par CI=Circuits Intégrés) ont été développés spécialement pour l'ATARI 520ST. L'ensemble de ces CI est géré par la GLUE (colle en anglais). Un autre composant est le MMU (Memory-Management-Unit). Cette unité gère la mémoire de travail, c'est-à-dire les 512 ou 1024 Koctets. Elle est même conçue pour gérer 4 Moctets (les versions plus récentes possèdent une RAM de 1 Mégaoctet).

Il faut ensuite mentionner le Shifter, qui s'occupe d'établir les images destinées au moniteur vidéo. Il interprète la mémoire-écran que l'image soit en couleur ou non, et envoie le signal vidéo correspondant au moniteur. Le port moniteur se trouve donc relié au Shifter. Ce port comporte une sortie pour le signal vidéo monochrome et trois sorties rouge, verte et bleue pour le signal RGB. L'utilisation simultanée d'un moniteur noir et blanc et d'un moniteur couleur est donc impossible ! L'ordinateur sait à quel type de moniteur il a affaire grâce à la sortie Monochrome-Detect, qui est soit à 5 volts (écran monochrome), soit à 0 volt (RGB).

Un autre composant important est le DMA-Controller (DMA=Direct Memory Access). Il accède directement à la mémoire et gère la transmission des données avec les unités de disquettes ou les disques durs. Les vitesses élevées auxquelles ont lieu ces transmissions, c'est à dire du 68000, exigeraient du CPU (Central Prozessor Unit), trop de travail. C'est pourquoi le DMA-Controller se charge de cette tâche.

Le pilotage des unités de disquettes est à la charge du Floppy-Disk-Controller WD 1772 qui, suivant les besoins, cherche les données sur disquette, fait démarrer le moteur, actionne la tête de lecture et prépare les données devant être transmises .Le contrôleur est capable de piloter des lecteurs de disquettes 'simple face' aussi bien que des lecteurs de disquettes 'double face'. Il n'y a donc aucun problème à relier l'ATARI ST à des lecteurs de disquettes double face. Le format (3,5 ou 5,25 pouces) ne joue également aucun rôle. La programmation du WD 1772 est très simple et si bien assisté par le système d'exploitation qu'il n'y a pas lieu de s'en occuper.

Un autre CI très important est le Multifunktionchip MPF 68901. Les entrée/sorties sont sa spécialité. C'est ainsi qu'il gère la sortie imprimante et les interfaces séries. Il s'occupe également de la fonction interne Time.

Il faut ensuite mentionner les 2 ACIAs (Asynchronous Communication Interface Adapter) qui gèrent la transmission série des données. Le premier s'occupe de la communication avec le clavier et le deuxième est responsable de l'interface MIDI. Cette dernière est intéressante, car elle permet de relier l'ordinateur à un réseau.

Ces composants sont naturellement programmables, comme les autres CI d'ailleurs. L'interface série, qui travaille relativement vite (31250 baud pour le système MIDI) est également utilisable pour bien d'autres applications.

Pour terminer, examinons encore le Sound-IC YM 2149 de l'ATARI ST. Ce CI fabriqué par la firme Yamaha renommée pour ses orgues électroniques et ses synthétiseurs, possède 3 générateurs de sons indépendants et un générateur de bruits. La courbe enveloppe, l'intensité, le choix et le mixage des différents canaux, ainsi que la valeur de la note sont autant de paramètres modifiables à l'aide de programmes. La largeur du spectre des fréquences dépasse de très loin le domaine audible. Il comprend des sons très graves et s'étend (presque) jusqu'aux ultra-sons.

Tous ces composants sont naturellement programmables. En ce qui concerne le Sound-chip, on peut le programmer très facilement grâce aux instructions BASIC : SOUND et WAVE. Pour les autres, c'est relativement plus compliqué. La plus simple des manipulations consiste à modifier les paramètres correspondants. En effet, chaque CI possède une zone mémoire réservée où sont stockées ses propres variables. C'est ici que s'effectue l'échange des informations (voir paragraphe 2.1). Mais restons encore pour le moment dans le domaine du hard, c'est-à-dire le domaine de la technique que l'on peut voir et toucher.

1.2 LES INTERFACES

La diversité des possibilités d'un ordinateur dépend en grande partie de ses capacités à communiquer avec l'extérieur. Cette communication est assurée par l'intermédiaire des interfaces, que nous allons maintenant examiner de plus près.

La transmission des données de l'émetteur au récepteur (collecteur) peut s'effectuer de 2 manières :

1. transmission parallèle
2. transmission série

1.2.1 Les Interfaces Parallèles

La transmission parallèle signifie qu'on émet d'un seul coup la totalité d'un mot mémoire. L'avantage de ce principe est la vitesse de transmission élevée pouvant ainsi être atteinte. L'inconvénient est le nombre élevé de connexions (un fil par bit).

Un exemple typique d'interface parallèle est le port (on dit aussi jonction ou sortie) Centronics. Il est en général utilisé pour l'imprimante, car il ne transmet des données que dans un sens, à savoir de l'ordinateur vers l'imprimante. Le port Centronics se situe à l'arrière du ST et porte l'indication 'Printer'.

La liaison entre ce port et l'imprimante est assurée par un câble comportant au moins 11 fils, dont la signification est la suivante:

Ordinateur Broche	Liaison		Imprimante Broche
1	Strobe	—>	1
2	Bit 0	—>	2
3	Bit 1	—>	3
4-9	Bit 2 à 7	—>	4-9
11	Busy	<—	11
18	Masse	—>	19

Comme on le voit, le signal BUSY se déplace dans le sens de l'imprimante vers ordinateur. Cette liaison retour appelée liaison 'Handshake' est nécessaire pour une bonne transmission des données.

Supposons que l'ordinateur veuille transmettre un caractère à l'imprimante. Il procède alors de la façon suivante : tout d'abord, il prépare la valeur ASCII du caractère et indique à l'imprimante qu'il est prêt à émettre, par un signal LO sur le fil Strobe. L'imprimante dépose alors cet octet dans sa propre mémoire. Si elle est en mesure de recevoir d'autres caractères, elle en fait part à l'ordinateur par un signal LO sur le fil BUSY. Celui-ci doit donc attendre que le signal BUSY soit retombé à LO.

Si l'imprimante se trouve être éteinte, alors l'ordinateur ne reçoit pas de réponse sur la liaison BUSY. Il attend encore un certain laps de temps, puis y renonce en affichant un message d'erreur à l'écran. Ce timeout (temps d'attente) est de 30 secondes sur l'ATARI ST, ce qui est relativement long. Cela vient du fait qu'il n'y a pas de signal ACK (Acknoledge = accusé de reception), par lequel l'ordinateur pourrait tout de suite se rendre compte que l'imprimante n'est pas prête. L'ATARI ST doit donc attendre un éventuel signal BUSY. Comme l'imprimante peut très bien être occupée, il faut lui laisser le cas échéant le temps de terminer ses activités momentanées.

L'ATARI ST offre une particularité liée à cette interface, qui est en réalité très peu courante. Il peut en effet collecter des informations à partir du port parallèle. Cela peut être très utile, car on a peut ainsi capter et exploiter un signal digital. La lecture des informations collectées sur le port imprimante se fait grâce à l'instruction INP(0) du langage de programmation BASIC, qui donne la valeur de l'octet qui s'y trouve. C'est ainsi que le ST peut recevoir des données qui sont en réalité destinées à une imprimante. Il suffit pour cela de relier le port Centronics d'un ordinateur avec le port parallèle de l'ATARI, en intervertissant les broches 1 et 11. On exécute ensuite à l'intérieur du ST le petit programme suivant :

```
10 rem ** simulation d'une imprimante **
20 x=inp(0)      : rem lecture de la donnée
30 print chr$(x): rem affichage du caractère correspondant
40 goto 20       : rem boucle sans fin
```

Toutes les données émises par d'autres ordinateurs et destinées aux imprimantes sont ainsi visualisées sur la sortie écran du ST. De cette manière, on peut capter des informations en provenance d'autres micros, sans utiliser le port RS-232.

Mais il y a mieux : on peut, grâce au port parallèle, entreprendre de piloter d'autres appareils. Les éléments de construction qui sont proposés par Fisher-Technik constituent un bel exemple. On peut ainsi réalisé des bras robotisés ou de simples tables traçantes qui sont commandés par l'ordinateur.

Le port parallèle est également très bien adapté au pilotage d'un ascenseur. En effet, pour cette tâche, l'ordinateur doit recevoir des données, les traiter, puis donner des ordres. On pourrait penser que tout cela n'est qu'un jeu, mais cela a de réelles applications dans la vie courante : en effet, tous les robots industriels fonctionnent sur ce principe. L'ATARI ST est donc théoriquement capable de piloter une machine outil.

1.2.2 Les Interfaces Séries

Dans la transmission série, un mot est transmis bit par bit. C'est pourquoi un seul fil de connexion suffit. Cependant, cette méthode est plus compliquée que la transmission parallèle.

L'interface série de l'ATARI ST porte l'inscription 'Modem' et son port est situé à l'arrière de l'appareil comporte 25 broches. Il s'agit d'une interface normalisée RS-232 (ou V24) qui travaille dans les 2 sens, contrairement au port Centronics. Les données sont transmises via la broche 2 (TD = Transmitted Data) par un signal HI/LO de + ou - 12 volts.

Cette prise est prévue pour le branchement d'un modem téléphonique, comme l'indique l'inscription 'Modem'. Un modem permet de relier l'ordinateur au réseau commuté (téléphonique), en transformant les signaux HI et LO en deux sonorités différentes. Si vous voulez transmettre des données (le plus souvent du texte) à quelqu'un qui possède également un modem, il vous suffit de l'appeler. Quand vous aurez chacun posé l'écouteur sur le modem, dans lequel se trouve un microphone et un haut-parleur, la transmission pourra commencée.

Pour signaler au collecteur (récepteur) qu'un mot va commencer, on émet au début de chaque mot un bit appelé bit Start. La ligne passe alors au niveau HI. Puis vient la transmission des données proprement dite qui se termine par un signal LO (bit Stop). La vitesse de transmission se mesure en baud, c'est à dire en bits par seconde (300 bauds pour un modem accoustique). La transmission est entièrement gérée par le système d'exploitation de l'ATARI ST, de telle sorte qu'une seule mise au point de l'interface est nécessaire.

Si vous ne connaissez personne qui possède un modem, vous pouvez également appeler des centres serveurs. Ces derniers peuvent être de nature privée ou publique. Vous trouverez leurs numéros de téléphone dans des périodiques spécialisés.

1.2.3 Le port des unités de disquettes

Le port se trouvant à l'arrière de l'ATARI et portant l'inscription 'Floppy' permet de relier un ou deux lecteurs de disquettes. Il est géré par le Disk-controller WD 1770, qui transforme les instructions du système d'exploitation en signaux. Si l'on demande par exemple le répertoire d'une disquette, le GEMDOS invite le contrôleur à lui communiquer les informations contenues sur le secteur correspondant de la disquette choisie. Le contrôleur exécute ensuite les tâches suivantes :

1. Selection du lecteur choisi (1 ou 2)
2. Mise en route du moteur qui fait tourner la disquette
3. Positionnement de la tête de lecture sur la piste qui contient le secteur demandé
4. Attente du signal Index-pulse qui survient à chaque tour et permet de connaître l'angle instantané de la disquette
5. Quand le secteur a été atteint, lecture et transformation du signal analogique délivré par la tête de lecture en un signal digital
6. Transmission de ce signal à l'ordinateur

A cela s'ajoute encore le choix de la face de la disquette (uniquement dans le cas d'un lecteur de disquettes double face). Il faut remarquer que le signal correspondant provient du Sound-Chip ! Le lecteur classique SF 354 avec 500 Koctets dans l'état non formaté ne lit et n'écrit que sur une face de la disquette. Cependant, les lecteurs pouvant travailler avec les 2 faces sont également entièrement gérés et directement connectables. Voici maintenant la significations des différentes broches :

Broche	Dénomination	Signification
1 (30)	Read Data	C'est ici qu'arrive le signal émit par la tête de lecture
2 (32)	Side 0 Select	Choix de la face de la disquette
3 (3)	GND	Masse (0 volt)
4 (8)	Index Pulse	Ce signal est émis à chaque tour de disquette
5 (10)	Drive 0 Select	Selection de l'unité 1
6 (12)	Drive 1 Select	Selection de l'unité 2
7 (33)	GND	Masse
8 (16)	Motor on	Mise en route du moteur
9 (18)	Direction in	Signal correspondant à la direction de déplacement de la tête de lecture
10 (20)	Step	Déplacer la tête de lecture d'un pas vers l'intérieur ou l'extérieur
11 (22)	Write Data	Ecrire les signaux émis par l'ordinateur
12 (24)	Write Gate	Signal autorisant l'écriture des données
13 (26)	Track 0	Signal provenant du lecteur et indiquant que la piste extérieure a été atteinte
14 (28)	Write Protect	Signal indiquant que la disquette est protégée à l'écriture

Ces signaux sont heureusement normalisés. La configuration du port permet de brancher n'importe quel lecteur de disquettes possédant une prise appelée Shugart. D'autres lecteurs peuvent donc être connectés sans aucune difficulté. Attention cependant : une mauvaise connexion pourrait détruire le contrôleur. Les numéros de broches de la prise Shugart correspondants sont indiqués entre parenthèses.

Avant la première utilisation d'une disquette, il faut la formatter. Dans le menu de formattage proposé par le Desktop, on voit que les lecteurs double face ont également été prévus. Avec des lecteurs simple face (uniquement déstinés à lire et écrire sur une seule face), il ne faut jamais choisir l'option 'double face'. L'inverse est cependant possible.

Le formatage d'une disquette consiste à effacer toutes les pistes et à mettre en place les secteurs. Toutes les informations qui pouvaient éventuellement s'y trouver sont ainsi détruites.

1.2.4 L'interface MIDI

A l'arrière de l'ATARI se trouvent encore 2 petites prises, qui attirent l'attention du fait qu'il s'agit de simples prises normalisés à 5 broches. Elles portent les inscriptions 'MIDI-IN' et 'MIDI-OUT' et permettent de relier des instruments de musique, comme par exemple des synthétiseurs. Comment cela fonctionne-t-il ? MIDI signifie Musical Instrument Digital Interface. Cette interface est également implantée dans beaucoup de synthétiseurs. Si l'on relie l'ordinateur avec un ou plusieurs (jusqu'à 16) orgues électroniques, le ST devient un véritable chef d'orchestre capable de mixer différentes mélodies, mais aussi de faire jouer lui-même les instruments. L'échange des informations s'effectue en série comme pour le port Modem, la seule différence étant la vitesse de transmission qui est ici de 31250 bauds (bits par seconde).

Comme plusieurs instruments peuvent être branchés simultanément, il faut que l'ATARI puisse selectionner l'appareil de son choix. Cela ne se passe pas de la même façon que pour la selection des lecteurs de disquettes (broche Select). Les récepteurs sont ici tous branchés en parallèle à l'interface MIDI, mais l'ordinateur émet d'abord un signal de selection, de telle sorte que seul l'instrument concerné prend en compte les instructions qui suivent.

Cette interface rapide semble très adaptée à d'autres applications. Ainsi par exemple, on pourrait relier ensemble plusieurs ordinateurs pour former un réseau (jusqu'à 17 ordinateurs). Il suffirait que chaque ordinateur possède une interface MIDI, et le ST pourrait communiquer avec chacun d'eux séparément. Ces échanges peuvent même être gérés à partir du BASIC, si les vitesses ne sont pas trop grandes. L'envoi d'une information se ferait grâce à l'instruction OUT 3,X et la lecture d'une information collectée grâce à X=INP(3). X représente ici le valeur de la donnée transmise.

1.2.5 Le port d'extension de la ROM

Sur le côté gauche de l'ordinateur se trouve encore une autre sortie. Elle ne porte aucune indication, mais est composée de 40 broches et permet l'insertion d'une carte électronique. Une telle carte peut comporter jusqu'à 128 Koctets de mémoires mortes (ROMs). Diverses applications sont possibles : extension du système d'exploitation, logiciels de secours pouvant relayer le système en cas de panne, ou tout simplement des jeux. Le grand nombre de broches s'explique facilement, car il faut pouvoir adresser cette zone mémoire supplémentaire et transmettre les données qui y sont contenues. 17 bits sont nécessaires à l'adressage des 128 K et 16 à la transmission des données . Apparemment, il manque une broche lecture/écriture, qui permettrait d'émettre des données. Cependant, il n'est pas prévu et même strictement interdit d'écrire à ces adresses. Vous pouvez cependant le vérifier, à condition toutefois de n'avoir rien d'important en mémoire principale. Le port d'extension se situe entre les adresses $FA 0000 et $FC 0000 (16 384 000 et 16 515 072). Si vous tapez par exemple POKE 16400000,0 , le système est réinitialisé et vous n'êtes plus dans le Desktop ! Toutes les données contenues dans mémoire BASIC sont ainsi détruites.

Le port d'extension est donc uniquement utilisable pour des cartes ROM. On appelle ces extensions des Cartridges, comme celles des anciens ATARI 400/600/800 (les anciennes cartes ne fonctionnent ici naturellement pas). Les premières adresses de ces cartes contiennent des variables, qui indiquent à l'ATARI de quel type de programme il s'agit. Si le premier mot long, situé à l'adresse $FA 0000, est égal à $FA52255F, alors il s'agit d'un programme de diagnostic. Si cette valeur est $ABCDEF42, c'est un programme d'application. Toutes les autres valeurs sont ignorées. Dans le cas d'un programme de diagnostic, le processeur, lors de l'initialisation du système, commence à exécuter les instructions à partir de l'adresse $FA 0004. Les programmes d'application, quant à eux, doivent contenir à partir de cette adresse diverses informations sur l'adresse de début, la longueur et le nom du programme.

1.2.6 Le port Souris/Joystick

Les 2 ports sur le côté droit de l'appareil sont prévus pour les Joysticks. Le sortie 0 peut également servir au branchement de la souris. La différence entre les Joysticks et la souris réside dans le fait que le Joystick possède 4 broches correspondants aux signaux de chacune des 4 directions ; si le Joystick est actionné dans une certaine direction, le niveau du signal correspondant devient LO (0 volt). La souris, quant à elle, transmet au système des impulsions , car elle peut être déplacée avec des vitesses différentes.

De ce fait, les prises du Joystick et de la souris sont un peu différentes. On peut brancher un Joystick sur n'importe lequel des 2 ports, mais la souris doit être branchée au port 0. Ce port possède, par ailleurs, quelques broches supplémentaires, car la souris dispose de deux boutons.

Ces deux ports sont uniquement utilisés en entrée. On ne peut pas émettre ici de signal vers l'extérieur, le processeur du clavier qui les gèrent ne connaissant pas d'instruction capable d'inverser le sens de transmission. Il y aurait certes la possibilité de remplacer ce processeur par un autre processeur dont le système d'exploitation connaittrait ces instructions supplémentaires, mais cela n'est pas réalisable en pratique. Les signaux de sortie peuvent d'ailleurs très facilement être émis grâce au port parallèle (Printer), tandis que ce port est très bien adapté à collecter des signaux d'entrée. Les tensions de signal permises sont 0 volt (LO) et 5 volts (HI) et correspondent à la norme TTL (TTL=Transistor-Transistor-Logik), qui est très employée en technique digitale. Si l'on veut par exemple connaître le niveau (LO ou HI) d'une broche de sortie de l'ordinateur, il suffit de la relier entre la masse (broche 8) et l'une des broches 1-4 de ce port, et d'interroger l'état du Joystick correspondant.

1.2.7 Le port moniteur

A quoi peut bien servir l'un des plus beaux ordinateurs, si on a pas instantanément une vision de son activité. Une sortie pour brancher un appareil capable de visualiser les informations est donc nécessaire, ce qui est bien entendu le cas pour l'ATARI ST.

Ce port porte l'inscription 'Monitor'. On peut non seulement y brancher un moniteur noir et blanc SM 124, mais également un moniteur couleur. Dans la configuration de ce port apparaissent les notions de Rouge, Bleu et Vert. Ces trois signaux constituent avec les 2 sorties de synchronisations horizontale et verticale le port appelé RGB et permettent à l'ATARI de gérer les couleurs, si l'on possède un moniteur couleur ou un téléviseur possédant une prise SCART.

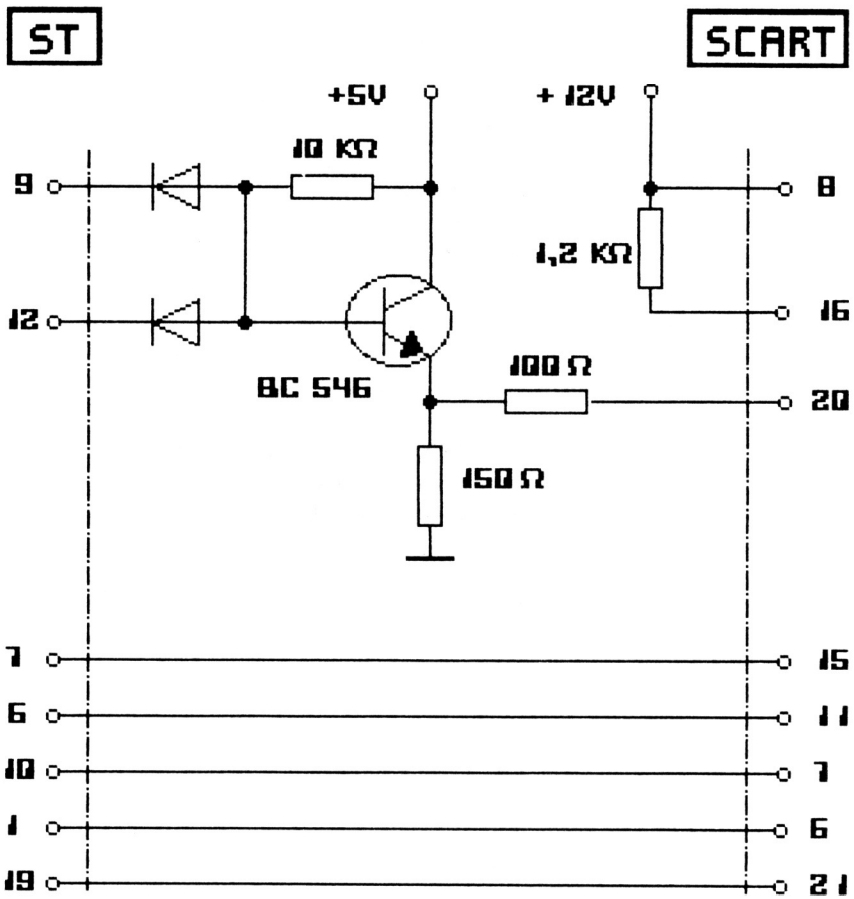
Il est vrai que le branchement de la prise d'un appareil de télévision ne s'effectue pas sans quelques problèmes, ce qui n'est pas le cas avec la 'curieuse' prise du moniteur, qui ne se trouve pas dans le commerce. De plus, les signaux émis par le ST ne sont pas directement utilisables par le recepteur TV. En effet, l'ATARI émet 2 signaux de synchronisation, alors que la prise SCART ne possède qu'une entrée synchro. Il faut donc les multiplexer (c'est à dire transmettre plusieurs signaux sur un seul fil). En outre, le téléviseur utilise 2 tensions différentes 2V et 12V pour transformer les signaux reçus en signaux RGB. Ces tensions ne sont pas disponibles sur le port moniteur. (Ceci ne concerne pas les nouvelles versions des ATARI 520 STF et 1040 STF).

On peut bricoler une prise provisoire, en utilisant une 'plaque à wrapper' sur laquelle on aura soudé des pattes faisant ainsi office de broche. Voici le tableau de correspondance des différentes broches :

ATARI ST		TV-PERITEL
1	Audio out	6 = Audio 2 à gauche
6	Vert	11
7	Rouge	15
10	Bleu	7
8	Masse	17
9	Hsync et	
12	Vsync	20 = Entrée video

Pour le multiplexage des signaux de synchronisation, il suffirait en fait de les relier à la broche 20 de la prise SCART par l'intermédiaire d'une résistance de 10 Kohms. Cependant, ce montage n'est pas très élégant et demande un débit important de la part de l'ATARI.

Pour obtenir une transformation correcte, il est nécessaire d'utiliser quelques composants électroniques : 1 transistor, 3 résistances et 2 diodes. Il faut de plus alimenter ce circuit avec une tension de 5V ; cette tension peut provisoirement être obtenue sur la broche 7 du port Joystick de l'ATARI ST. Le circuit peut être soudé sur la même plaque à Wrapper. Voici le plan de ce circuit :



Cependant, ces dépenses ne sont nécessaires que dans le cas de l'ATARI 520ST. En ce qui concerne son grand frère, le 1040 STF, il contient déjà ce circuit de multiplexage ; la configuration du port moniteur est donc de ce fait un peu différente. C'est ainsi que la broche 8 n'est pas reliée à la masse, mais à l'alimentation 12V qui est nécessaire quand on utilise un téléviseur comme sortie vidéo. Cette broche doit dans ce cas être reliée à la broche 8 de la prise SCART. On passe alors automatiquement en mode vidéo, dès que l'on branche le moniteur. Cette alimentation de 12V permet encore d'obtenir la tension de 2V indispensable grâce à 2 résistances, de telle sorte qu'une prise supplémentaire n'est plus indispensable pour relier le ST à un téléviseur à la norme RGB.

Sur le 1040 STF, la broche du port moniteur qui est reliée à la masse est la broche numéro 13 ; c'est donc la seule broche qui malheureusement ne correspond pas avec le montage précédent. Il faut donc avoir recours à un artifice pour replacer la patte à la bonne place.

Il suffit ensuite de modifier la place de la broche syncro, en l'ajustant sur la broche 2 de l'ordinateur. C'est ici en effet qu'est émis le signal mutiplexé de l'image RGB.

Cependant, je fus confronté à un autre problème. L'image était effectivement en couleur, mais, malgré mes efforts de persuasion, je n'arrivais absolument pas à obtenir de balayage vertical. Un coup d'oeil à l'adresse mémoire $448=1096 m'apporta la solution : le système d'exploitation croyait avoir affaire à un moniteur américain normalisé NTSC. Le problème provenait du fait que ce type d'appareil affiche 60 images par seconde, alors que notre norme PAL/SECAM a une fréquence de 50 Herz.

Pour corriger ce défaut, nous allons utiliser l'astuce suivante. La variable indiquant au système d'exploitation de quelle norme il s'agit (NTSC ou PAL) se trouve à l'adresse $5056. Si l'on modifie son contenu (initialement $67) en le remplaçant par la valeur $66 et que l'on réinitialise le système, on obtient, après avoir chargé le TOS.IMG, l'image couleur souhaitée.

Hélas, il n'est pas possible de modifier l'octet situé à l'adresse $5056 à partir du BASIC, mais uniquement à partir du logiciel utilitaire SID du kit de développement SID, car l'instruction POKE est inutilisable pour cette adresse. Enfin, il faut sauvegarder cette modification avec le système d'exploitation, afin de la recharger lors d'un RESET.

Si l'on ne possède pas le logiciel SID, on peut utiliser une autre astuce. En effet, on peut, par un BSAVE, sauvegarder un ensemble de données.Cet ensemble peut en particulier être un programme, directement exécutable. C'est exactement ce que nous allons faire maintenant. Ce programme génère du code machine (programme en langage machine) et le sauvegarde sur disquette. Le programme machine peut être appelé à partir du Desktop et permet de changer la valeur de la fréquence verticale de la sortie couleur, la nouvelle valeur correspondant à la norme PAL : 50 Herz. Le nom du programme machine SETPAL.PRG est donc clair, mais il peut être avantageusement remplacé par AASETPAL.PRG, car, de cette manière, le programme se situe en première place dans le repertoire. En considérant le fait qu'à l'appel de cette routine, l'image est plutôt perturbée, il sera ainsi plus simple de l'atteindre avec la souris !

```
10  rem ** Création de SETPAL.PRG **
20  for i=1 to 74 step 2        : rem boucle pour 36 mots
30  read a                      : rem lire le mot
40  poke i+1999,a               : rem le stocker en mémoire
50  next i                      : rem fin de la boucle
60  bsave 'SETPAL.PRG',2000,74 : rem sauvegarder
70  end
90  rem — données du programme-code —
100 data 24602,0,44,0,0,0,0,0,0,0,0,0,0,0
110 data 17063,16188,32,20033,-8196,0,6
120 data 5116,2,-1,-32246
130 data 12032,16188,32,20033,-8196,0,6
140 data 16999,20033,0,4,0,0
```

Avant d'exécuter le programme, n'oubliez pas de vérifier que vous ne vous êtes pas trompé en recopiant les données. Une toute petite erreur entraînerait à coup sûr le 'plantage' de l'appareil.

Le programme en langage machine se compose comme suit :

Les données de la ligne 100 représente l'en-tête du programme, qui est nécessaire à l'exécution de celui-ci. Elle comprend plusieurs indications : longueur et forme de l'ensemble du programme ; cela ne représente pas ici un grand intérêt pour nous. La ligne 110 met le processeur en mode Superviseur, dans lequel il est indispensable de se placer pour effectuer ces modifications. Puis, on pénètre dans la zone-mémoire réservée aux I/O (Input/Output = Entrées/Sorties), où est déterminée la fréquence verticale. Enfin, on replace le système en mode Utilisateur et on retourne au Desktop. Comment tout cela est-il compris par le processeur ? Nous le verrons plus tard dans la partie consacrée au langage-machine du MC 68000. Pour l'instant, continuons à observer l'architecture du ST.

A l'aide de cet exemple, nous pouvons, dès à présent, examiner une autre possibilité offerte par l'ATARI ST. Pour exécuter automatiquement notre programme-exemple dès la mise en route de l'ordinateur, il suffit de rajouter un autre répertoire sur notre disquette système. Ce répertoire doit porter le mot AUTO et provoque, après le chargement du système d'exploitation, l'exécution de tous les programmes s'y trouvant. On peut également y inclure l'interprèteur BASIC.

1.3 UN CLAVIER INTELLIGENT

Comme je l'ai déjà indiqué, le processeur implanté dans le clavier a de multiples tâches. Il décharge dans une large mesure le MC 68000 de la gestion du clavier et de la souris. Grâce à une mémoire qui lui est propre, le clavier est en mesure de connaître en permanence l'état des touches, de la souris, des joysticks et même de l'heure. Toutes ces informations sont communiquées au calculateur principal si ce dernier le demande. Le calculateur central émet alors en série un numéro de commande, éventuellement suivie de quelques paramètres, vers le clavier ; ce dernier renvoie ensuite au MC 68000 les informations voulues. Tout cela s'effectue à la vitesse de 7812.5 bits par seconde. Le processeur du clavier est programmé de telle sorte qu'il comprend les instructions suivantes :

1.3.1 L'ensemble des commandes

Commande	Effet
7	mise en place de la gestion des touches de la souris
8	dès maintenant, signaler les positions relatives de la souris
9	dès maintenant, signaler les positions absolues de la souris
10	signaler les mouvements de la souris comme s'il s'agissait d'actions sur les touche-curseurs
11	mettre en place le seuil de déplacement au delà duquel les mouvements de la souris sont pris en compte
12	mettre en place l'échelle des positions de la souris
13	demande de la position absolue de la souris
14	mise en place du compteur interne de coordonnées
15	l'origine de l'axe Y des ordonnées se trouve en bas
16	' ' ' ' en haut
17	reprendre la transmission des données (après 19)
18	déconnecter la souris
19	arrêter la transmission des données
20	dès maintenant, signaler chaque mouvement du joystick
21	arrêt de la fonction 20 et déconnexion de la souris
22	interrogation de la position du joystick
23	mettre en place la durée entre les émissions d'information venant du joystick
24	mettre en place l'intervalle du temps au delà duquel on prend en compte une nouvelle pression sur la touche de tir
25	dès maintenant, signaler les mouvements du joystick comme s'il s'agissait d'action sur les touche-curseurs (joystick 0)
26	déconnecter le joystick
27	mettre en place l'horloge
28	demande de l'heure
32	modifier la mémoire du clavier
33	lire la mémoire du clavier
34	exécution d'un programme contenu dans la mémoire du clavier

A part cela, il répond à diverses demandes sur l'état des fonctions (on/off).

Mais examinons à présent quelques exemples d'utilisation des instructions destinées au clavier. La difficulté de cette programmation provient du nombre variable de paramètres et de leurs significations qui dépendent de chaque fonction. De plus, les résultats des fonctions ne sont pas toujours faciles à obtenir. Pour s'en rendre compte, prenons un exemple. La fonction 22 demande au processeur du clavier l'état instantané du joystick 1. La réponse est obtenue à l'adresse 2633. Un programme permettant de connaître l'état du joystick a donc l'allure suivante :

1.3.2 L'interrogation de la position du joystick

```
10 rem ** interrogation de l'état du joystick 1 **
20 out 4,22                    : rem appel de la fonction
25 DEFSEG = 1
30 js = peek(3592)             : rem lire le résultat
```

Grâce à l'instruction OUT 4,n (où n est le numéro de commande) du BASIC de l'ATARI, on demande au clavier d'interroger l'état du joystick 1 et de transmettre la valeur correspondante au calculateur principal. Le système d'exploitation dépose ensuite cette valeur à l'adresse 3592. Puis on la lit grâce à PEEK(3592). Le résultat est sous forme binaire : les bits 0 et 1 indiquent la position verticale et les bits 2 et 3 la position horizontale. Comme on le voit, il n'est pas nécessaire de transmettre des paramètres avec cette commande. Voyons maintenant un exemple de fonction avec paramètres :

1.3.3 Le pilotage du curseur avec la souris

```
10 rem ** la souris remplace les touche-curseurs **
20 out 4,10                    : rem envoyer la commande
30 out 4,10                    : rem distance suivant X
40 out 4,15                    : rem distance suivant Y
```

On demande ici au clavier d'interprèter les déplacements de la souris comme s'il s'agissait d'action sur les touche-curseurs. Cette méthode est très utile pour la mise au point de programmes BASIC, car on peut ainsi déplacer très vite le curseur vers la droite. Les valeurs des distances qui sont données dans les lignes 30 et 40 correspondent aux distances de déplacement horizontale et verticale de la souris pour un déplacement unitaire du curseur sur l'écran. Il n'est donc pas conseillé ici de prendre des valeurs élévées, si l'on possède une table de petite taille. Pour considérer à nouveau la souris comme index, il faut se servir de l'instruction OUT 4,8. Comme on peut le voir, il faut faire très attention de respecter le bon nombre de paramètres, sinon il se pourrait qu'une valeur supposée être un paramètre ait par exemple pour effet de stopper l'échange de données entre le clavier et l'ordinateur.

1.3.4 La gestion de la date et de l'heure

Examinons à présent un exemple dans lequel des paramètres sont transmis à la fois en entrée et en sortie. Il s'agit de la gestion de l'horloge intégrée dans le clavier, qui peut être très intéressante dans certaines applications.

```
10 rem ** affichage de l'heure et de la date **
20 out 4,28                    : rem demander l'heure
30 for i=2626 to 2631          : rem zone tampon (buffer)
                                 contenant l'heure et la date
40 x = peek(i) and 255         : rem lire deux chiffres
                                 (contenus dans un seul octet)
50 a$=a$ + str$(int(x/16)) : rem premier chiffre
60 a$=a$ + right$(str$(x and 15),1)
70 a$=a$ + '.'
80 next i
90 print 'Date  : ' ; left$(a$,12)
100 print 'Heure : ' ; rights(a$,12)
110 end


130 rem ** initialisation de l'heure et de la date **
140 out 4,27        : rem commande Set-time
150 print'entrez la date et l'heure (AAJJMMHHMMSS) : '
```

```
160 input a$        : rem saisie
170 for i=1 to 6    : rem calcul du code BCD
180 d=0
190 d=d + asc(mid$(a$,i*2,1)) - 48
200 d=d + (asc(mid$(a$,i*2-1,1))-48)*16
210 out 4,d         : rem envoi des paramètres
220 next i
230 a$="": goto 10 : rem affichage des nouvelles valeurs
```

Il s'agit en fait de deux programmes indépendants. L'exécution de la première partie entraîne l'affichage de la date et de l'heure dans l'ordre suivant : année, jour, mois, heure, minute, seconde. Le format de la chaîne de caractères a$ est entièrement laissé à votre actif. D'autre part, on peut transformer cette partie en un sous-programme en remplacement dans la ligne 110 le END par un RETURN. L'étape la plus délicate de l'exploitation des données reçues est la conversion de la forme BCD (Binary Coded Decimal) en chiffres séparés. Dans ce codage, chaque chiffre n'occupe en effet que 4 bits d'un octet (8 bits). C'est ainsi que, par exemple, la valeur du nombre 31 en BCD (0011 0001 en binaire) est égale à la valeur hexadécimale $31, ce qui correspond à 49 en décimal (voir aussi chapître 2.1).

Ce même problème apparait également lors de la transformation inverse : chiffres séparés en nombres BCD. La routine qui initialise la date et l'heure commence à la ligne 130. Après l'envoi de la commande 'Set-time', on saisit la date et l'heure dans l'ordre indiqué ci-dessus. Par ex. : 053010192015 pour le 30/10/85 à 19 h 20 mn 15'. Le chiffre 8 de l'année est automatiquement rajouté. Après la saisie, chaque chiffre, converti en nombre BCD, est envoyé au processeur du clavier. L'instruction GOTO 10 sert uniquement à contrôler le bon fonctionnement du programme et peut être supprimée.

1.3.5 La scrutation du clavier

Nous allons encore examiner une fonction du clavier qui n'est pas complètement inutile, puisqu'il s'agit de gestion des touches elle-même. L'ensemble des touches est supervisé en permanence par le processeur implanté dans le clavier. Si une touche est enfoncée, ce dernier stocke la valeur de cette touche dans sa mémoire propre.

Cette mémoire a une taille relativement petite, 128 Koctets, mais qui suffit largement ; elle contient, outre les valeurs des touches enfoncées, certaines informations supplémentaires. Le calculateur principal ou le système d'exploitation prend régulièrement connaissance de ces valeurs. Les touches enfoncées sont ensuites stockées en mémoire principale et lues si besoin.

Si vous voulez connaître l'état d'une touche à partir d'un programme BASIC, plusieurs méthodes sont possibles. On peut attendre qu'une touche soit enfoncée avec l'instruction X=INPUT$(1) et exploiter la valeur X par la suite. Mais cette instruction ne donne que le code ASCII d'une touche alphanumérique normale. Par contre, les informations obtenues grâce à l'instruction X=INP(2) sont bien plus étendues. La valeur X ainsi obtenue correspond au code de n'importe quelle touche enfoncée. Les touches fonctions sont donc également prises en compte. Leur code est compris entre 187 et 196. Pour attendre qu'une touche fonction soit enfoncée, on peut utiliser le programme suivant :

```
100 print 'Appuyer sur une touche fonction, s.v.p. ! '
110 x = inp(2)                    : rem lire la valeur de la
                                    touche enfoncée
120 if x<187 or x>196 then 110  : rem mauvaise touche
130 ft = x - 186                  : rem numéro de fonction
140 on ft goto f1,f2,f3,f4,f5,f6,f7,f8,f9,f10
```

L'action sur une touche-fonction a pour effet d'exécuter la partie du programme souhaitée, qui peut bien entendu porter un autre nom que f1-f10. Cette méthode de programmation est très utilisée pour la gestion des menus.

1.4 LA SOURIS

La souris grise avec ses deux touches constitue un formidable périphérique d'entrée. Le va-et-vient continuel du regard entre l'écran et le clavier est ici supprimé, car on 'tape' pour ainsi dire à l'écran, du moins en ce qui concerne le choix des menus. A chaque mouvement de la main correspond un déplacement à l'écran suivant X et Y, avec un rapport de réduction de 4 pas par millimètre. Comment cela est-il possible ?

Un coup d'oeil à l'intérieur de la souris nous autorise déjà à formuler quelques suppositions. Si on sort la bille, en faisant coulisser le fond de la souris, on aperçoit 3 petits rouleaux en métal. Le rouleau oblique sert uniquement à stabiliser la bille ; les deux autres communiquent leurs mouvements rotatifs à deux autres disques qui contiennent chacun deux portes optiques.

Le processeur HDV 6301 V1, qui se trouve dans le clavier, exploite les 4 séries d'impulsions. Supposons que la souris soit déplacée vers l'avant, c'est-à-dire suivant la direction Y. Le processeur reçoit sur les entrées 'Y-Puls' deux signaux déphasés. Il attend dans un premier temps que les deux signaux ait la valeur logique 1, c'est à dire le niveau HI. Il en déduit le sens de déplacement, sachant que celui-ci correspond au premier des deux signaux qui retombe à 0 (LO).Le même principe s'applique pour le déplacement suivant X. En combinant les deux, on peut donc connaître la direction réelle du mouvement de la souris.

Si le système le demande, le processeur du clavier peut lui transmettre aussi bien la direction de déplacement de la souris, que les valeurs des touches enfoncées, les positions des joysticks, l'heure ou la date. Cette transmission s'effectue à la vitesse de 7800 bits par seconde. Le programmeur n'a heureusement pas à s'occuper de cette transmission. Il peut, en ce qui concerne l'interrogation de l'état de la souris, directement utiliser certains langages de programmation spécialisés. Le DR LOGO fourni avec l'ordinateur offre en particulier l'instruction MOUSE, spécifique au langage LOGO, qui retourne une liste de paramètres.

1.4.1 La souris comme crayon à dessin

Un programme permettant de dessiner sur l'écran (fenêtre graphique) a alors l'allure suivante :

```
TO DRAW
 IF (ITEM 4 MOUSE) [SETPOS MOUSE]
 DRAW
 END
```

Ce programme est entré dans l'éditeur LOGO et est appelé par l'ordre DRAW. Il s'agit d'un programme récursif, c'est à dire qui s'appelle lui-même.

Cela vient du fait que l'instruction DRAW apparaît en fin de procédure. La deuxième ligne montre l'emploi de l'instruction MOUSE du langage LOGO. Le plus simple est de la considérer comme un ensemble de variables, dont les valeurs sont contenues dans une liste.
Cette liste est composée comme suit :
[x y b1 b2 b3] ; x et y sont les coordonnées momentanées du curseur ; b1 et b2 représentent l'état des deux touches de la souris, b1 pour la touche gauche et b2 pour la touche droite. Ces variables sont vraies (TRUE) quand la touche correspondante est enfoncée ; enfin, b3 indique si le curseur de la souris se situe dans la fenêtre graphique ou non (TRUE si oui). Dans le programme ci-dessus, on teste le quatrième élément de la liste, c'est à dire l'état de la touche droite, grâce à l'instruction (ITEM 4 MOUSE). Si la touche est enfoncée, on trace une droite jusqu'au curseur de la souris.

En exécutant le programme, vous vous rendrez sûrement compte que rien ne se passe , si la souris est déplacée pendant que l'on appuie sur la touche. Ceci est également vérifiable lorsqu'on exécute des programmes BASIC. La raison à cela vient du GEM. Celui-ci, devant constamment afficher le curseur, mobilise totalement le microprocesseur. Cet effet parasite souvent indésirable, peut s'avérer utile. En effet, on peut ainsi interrompre provisoirement le déroulement d'un programme pour lire les données intermédiaires par exemple.

Le DR LOGO est un langage très adapté à ce type de programmation. Tous les langages n'offrent pas une aussi grande facilité. Le langage préféré de tous, le BASIC, n'offre pas d'instruction susceptible d'interroger l'état de la souris.
Il faut donc le faire, non sans quelques efforts, en utilisant directement le VDI du GEM (voir aussi paragraphe 3.2 : le GEM).

Une remarque concernant les possesseurs d'un ancien modèle ATARI ou un ordinateur personnel du même type : la prise de la souris peut s'adapter à n'importe quel port joystick. Il suffit pour cela d' écrire un petit programme machine adapté et rapide. Vous pourrez ainsi brancher la souris et l'utiliser de la façon qui vous semblera judicieuse. La lecture des informations collectées doit avoir lieu environ 1000 fois par seconde : cela permet de prendre en compte des déplacements de l'ordre du 1/4 de millimètre.

2. LES MEMOIRES DE DONNEES

Les données et les programmes doivent naturellement être stockés d'une manière ou d'une autre. Techniquement parlant, il y a différents moyens , comme par exemple les semi-conducteurs et les disques magnétiques. Nous allons maintenant examiner ces différentes 'armoires de données' et leurs avantages respectifs.

2.1 LA MEMOIRE INTERNE

La mémoire interne peut être divisée en deux parties :

- la mémoire vive (RAM = Random Access Memory)
- la mémoire morte (ROM = Read Only Memory)

L'ATARI 520ST et le 260ST possèdent une RAM de 512 Koctets, 524288 octets exactement. L'ATARI 1040 ST en possède le double, à savoir 1024 Koctets. Ces mémoires vives sont constituées de composants électroniques qui ne conservent donc leur contenu que si on les alimente avec un courant suffisant. Si une coupure de courant intervient, la totalité du contenu de la mémoire est perdue. Pour se préserver de cet incident peu fréquent, mais d'autant plus désagréable, il est conseillé d'effectuer de temps à autre des sauvegardes sur disquette.

Dans cette gigantesque mémoire se trouvent donc toutes les données et programmes nécessaires au bon fonctionnement de l'ordinateur.

Dans la version de l'ATARI ST qui est étudiée ici, le système d'exploitation de l'ordinateur est également chargé en RAM et occupe 200 Koctets. A cela s'ajoute encore la mémoire écran qui occupe 32 Koctets, ainsi que certaines variables et tables indispensables au système d'exploitation. Si maintenant, on charge encore le langage de programmation LOGO par exemple, dont la taille est de 110 Koctets, on s'aperçoit qu'il ne reste plus grand chose de libre.

Comme son nom l'indique, la RAM est aisément modifiable. Les données utilisées par le système d'exploitation peuvent donc être manipulées, ce qui constitue une véritable 'provocation' pour les programmeurs qui peuvent ainsi très facilement exploiter à fond tout le système. Les instructions PEEK et POKE sont prévues à cet effet, mais avant de les utiliser, il nous faut connaître exactement ce que contiennent les différentes adresses.

Remarque :

La manipulation du contenu d'une adresse inconnue peut avoir des effets totalement inattendus et souvent facheux, dont on ne se rend compte parfois que plus tard !

Avant de passer aux essais, nous allons d'abord examiner plus en détail le contenu de l'ATARI ST.

Après la mémoire vive, nous allons parler de la mémoire dont le contenu n'est pas modifiable : la ROM. Elle est constituée d'éléments dont le contenu ne s'efface pas, même sans alimentation. Dans les nouvelles versions de l'ATARI ST, le système d'exploitation TOS sera également implanté en ROM. De cette façon, il sera utilisable dès la mise sous tension. Dans l'ordinateur qui nous intéresse, la ROM contient uniquement un programme qui charge le TOS à partir du lecteur de disquette. Le seul avantage de cette méthode réside dans la possibilité de manipuler le système d'exploitation.

Examinons maintenant la totalité de la zone mémoire de l'ATARI ST. Comme cet ordinateur possède un bus d'adresse de 24 bits, il peut donc adresser directement 2^24 octets = 16 Mégaoctets.

Cette gigantesque zone adressable n'est en réalité pas entièrement recouverte de composants électroniques. Le schéma suivant montre la configuration de la mémoire principale :

2.1.1 Configuration de la mémoire principale de l'ATARI ST

00 0000	ROM	Reset : pointeur de la pile superviseur
00 0004	ROM	Reset : compteur ordinal initial
00 0008	RAM	Début de la RAM
	.	
	.	
07 FFFF	RAM	Fin de la RAM avec 512 Koctets
	.	
0F FFFF	RAM	Fin de la RAM avec 1 Moctet (1040 STF)
	.	
1F FFFF	RAM	Fin de la RAM avec 2 Moctets
	.	
3F FFFF	RAM	Fin de la RAM avec 4 Moctets (Maximum)
40 0000	unused	Champ non utilisé
F9 FFFF	.	
FA 0000	ROM	Extension de la ROM (128 Ko)
FB FFFF	.	
FC 0000	ROM	ROM : Système d'exploitation (192 Ko)
	.	
FE FFFF	.	
FF 0000	unused	Champ non utilisé
FF 7FFF	.	
FF 8000	I/O	Domaine des I/O (2 Ko)
FF 8800	.	
FF A000	I/O	Domaine des I/O (2 Ko)
FF BFFF	.	
FF C000	unused	Champ non utilisé
FF FFFF	.	

Comme vous pouvez le voir, la zone située entre l'adresse 00 0000 et l'adresse 07 FFFF correspond à la RAM. Une extension jusqu'à 4 Mégaoctets est autorisée par le système d'exploitation et ne devrait pas poser de problème à un bricoleur. Le système d'exploitation sera lui-même (plus tard) implanté dans la ROM qui commence à l'adresse FC 0000. Dans la partie 'haute' de la mémoire se trouvent les zones réservées aux I/O (cf paragraphe 1.1). C'est ici que sont échangés les paramètres avec les autres processeurs, le micro-processeur ayant un accès privilégié. Une intervention de l'utilisateur à l'intérieur de cette zone est immédiatement sanctionné par une interruption du programme en cours, appelé BUS-ERROR, qui entraîne l'affichage de deux 'champignons atomiques' et le retour au Desktop du GEM. La zone réservée au I/O est composée de la manière suivante :

2.1.2 Adresses des I/O

à partir de	FF 8000	configuration de la mémoire
' '	FF 8200	registres pour le Video-chip
' '	FF 8600	DMA/DISK-Controller
' '	FF 8800	registres pour le Sound-chip
' '	FF FA00	Timer-chip et Interrupt-chip
' '	FF FC00	Processeur du clavier et MIDI-chip

Il faut encore mentionner une autre zone mémoire importante : le 1er Koctet à partir de l'adresse 0. C'est ici que se trouvent les paramètres nécessaires au système. A cause de leur importance, l'utilisateur n'y a pas non plus accès (sous peine de voir à nouveau les fameux champignons atomiques). Il s'agit de paramètres appelés pointeurs. Ces pointeurs contiennent eux-même d'autres adresses, appelées adresses de saut, auxquelles se branche le processeur après une interruption. C'est ainsi que, par exemple, la violation des zones mémoires protégées (les I/O mais aussi les tables de pointeurs) provoque de la part du microprocesseur l'utilisation de ces pointeurs. La table des pointeurs est organisée de la façon suivante :

2.1.3 Les pointeur-erreurs

Nr.	Adresse	utilisé lors d'un(e)
	$ 0	Reset: SSP initial
	$ 4	Reset: PC initial
2	$ 8	Bus-error (cf ci-dessus)
3	$ C	Adress-error
4	$ 10	Instruction illégale
5	$ 14	Division par zéro
6	$ 18	Instruction CHK
7	$ 1C	Instruction TRAPV
8	$ 20	Violation des protections
9	$ 24	Trace
10	$ 28	Instructions Axxx de l'émulateur
11	$ 2C	Instructions Fxxx de l'émulateur
	$ 30-$ 38	Reservé
	$ 3C	Interruption non initialisée
	$ 40-$ 5F	Reservé
	$ 60	Interruption non autorisée
	$ 64-$ 7F	Interruption de niveau 1-7
	$ 80-$ BF	Instruction TRAP
	$ C0-$ FF	Reservé
	$100-$3FF	Table des interruptions :
	$100	Port parallèle interne
	$104	RS232 Carrier Detect
	$108	RS232 Clear to send
	$10C	inutilisé
	$110	inutilisé
	$114	Horloge-système 200 Hz
	$118	Interruption clavier/MIDI
	$11C	inutilisé
	$120	HSync
	$124	Erreur d'émission sur le port RS232
	$128	Buffer d'émission du RS232 vide
	$12C	Erreur de reception sur le port RS232
	$130	Buffer de reception du RS232 plein

Tous ces pointeurs ne sont pas utilisés uniquement lors d'une erreur, mais beaucoup provoquent une interruption souvent désagréable pour le programme en cours. Le numéro des pointeurs correspond au nombre de champignons atomiques qui apparaissent sur l'écran, après une interruption du système, ce qui permet donc de connaître la nature de l'erreur. Si une telle interruption a lieu, le système d'exploitation essaie de sauver ce qui peut être sauver. Souvent cela se traduit par un échec, et il faut à nouveau avoir recours à la touche RESET. Si cependant il est possible de reprendre l'exécution du programme interrompu, l'ordinateur a besoin des informations sur l'état du système avant l'apparition de l'erreur. Ces informations sont stockées, avant le 'plantage' du système, aux adresses suivantes :

$380	=$12345678, si les données sont valides
$384	à partir de cette adresse, se trouvent les registres D0- D7 qui ont pûs être sauvés
$3A4	et à partir de cette adresse, les registres A0-A6
$3C0	ancien pointeur de la pile-superviseur A7
$3C4	contient le numéro de l'erreur qui a provoqué l'interruption
$3C8	ancien pointeur de la pile-utilisateur
$3CC	16 mots de l'ancienne pile-superviseur

La signification des pointeurs et de leur utilisation vous apparaîtront peut-être inconnues, car il s'agit ici du domaine du langage machine. Pour mieux comprendre ces processus et leurs utilisations ,je vous conseille le livre intitulé **"LA BIBLE DE L'ATARI ST"** également distribué par **MICRO APPLICATION.** Mais la question primordiale reste posée : Que représentent au juste ces pointeurs ? Voyons cela de plus près.

2.1.4 Les pointeurs

Les pointeurs sont des éléments très importants du software en général. Un pointeur est une adresse à laquelle se trouve une autre adresse. Voici un exemple : la variable système $44E. A cette adresse, on trouve une autre adresse, celle de la mémoire écran, qui se situe dans le haut de la mémoire principale. Le BIOS, ainsi que le GEM ne connait l'emplacement de la mémoire écran qu'en faisant appel à ce pointeur.

La mémoire écran de l'ATARI ST avec 512 Ko n'est d'ailleurs pas à la même place que celle de l'ATARI 1040 STF avec 1 Mo. Cette observation montre justement la nécessité de l'emploi des pointeurs. Un seul et même programme peut donc, par l'intermédiaire d'un pointeur, adresser une zone-mémoire située en réalité à des adresses différentes ; seul la valeur du pointeur change.

Reprenons l'exemple du pointeur de la mémoire écran. Pour accéder à celle-ci, il nous suffit de lire la valeur du pointeur et de l'utiliser pour l'adressage.

```
10 rem ** affichage de points à l'écran par POKE **
20 adresse =peek(1102) *65536 +peek(1104)
25 if peek(1104)<0 then adresse =adresse +65536
30 l = 20                          : rem numéro de la ligne
40 c = 5                           : rem numéro de la colonne
50 nol = peek(9598)                : rem nombre d'octets par
                                     ligne
60 poke adresse + c + l*nol,255    : rem afficher le trait
```

Les lignes 20 et 25 montrent la difficulté de l'emploi des pointeurs. La zone adressable de l'ATARI ST dépassant de beaucoup 64 Koctets, un pointeur doit avoir plus de 16 bits. Il s'agit en effet d'adresses de 32 bits, que l'on appelle parfois des mots doubles (ou mots longs), dont l'évaluation en BASIC est laissée à la charge du programmeur. Dans l'exemple ci-dessus, le premier mot (mot de poids fort) du pointeur se trouve à partir de l'adresse 1102 et doit donc être multiplié par 65536, c'est à dire 2^16. Puis, on ajoute la valeur du mot de poids faible pour obtenir la valeur de l'adresse pointée.

Le système d'exploitation n'a pas besoin d'effectuer ces opérations, car le microprocesseur peut travailler avec des mots doubles. Par contre, le calcul de l'adresse correspondant à l'emplacement souhaité sur l'écran est effectué par le calculateur central de la mème façon que dans les lignes 50 et 60. Un caractère n'étant finalement rien d'autre qu'un motif contenu dans rectangle de 16 * 8 points, l'affichage d'un caractère à l'écran peut également se réaliser de cette manière. A cet effet, modifions légèrement le programme précédent :

```
60  for i=0 to 10                    : rem début de la boucle
70  read x$                          : rem lire une ligne du motif
80  x=0 : for j=1 to 10              : rem évaluation
90  x= x-(mid$(x$,j,1)='*') * 2^ (10-j)
100 next j
110 poke adresse+c+(l+i)*nol,x       : rem affichage de la ligne du
                                       motif correspondante
120 next i                           : rem et ainsi de suite
130 end
135 rem — données du modèle —
140 data '          '
150 data '   **     '
160 data '  ****    '
170 data ' ******   '
180 data ' ** ** ** '
190 data '**********'
200 data '** **** **'
210 data '**  **  **'
220 data ' **    **  '
230 data '  ******   '
240 data '   **     '
```

Nous avons donc contourné le système d'exploitation pour afficher, comme par magie, un petit sourire à l'écran. Le système d'exploitation aurait agi de la même façon et aurait également utilisé ce pointeur. Mais que se passe-t-il si on change la valeur du pointeur ? Essayons donc de le modifier en entrant :

poke 1104,peek(1104)+1600 (Return)

L'image de l'écran fait un saut vers le bas. On peut s'en rendre compte en fixant des yeux n'importe quelle fenêtre. Le système d'exploitation est induit en erreur par la fausse valeur du pointeur et utilise une mauvaise zone mémoire pour la mémoire écran ! Replaçons cependant le système dans son état initial pour éviter des effets secondaires désagréables.

poke 1104,peek(1104)-1600 (Return)

Et tout rentre dans l'ordre. Mais il y a également beaucoup de pointeurs, dont la modification cause bien plus de dégâts que pour le pointeur de la mémoire écran. C'est en particulier le cas pour les pointeurs de routines.

Ces pointeurs représentent des adresses de saut (ils pointent sur des routines machines) où se branche le microprocesseur. Une utilisation très importante de ces pointeurs de saut sont les pointeurs d'interruption, qui entrent en jeu pour diverses raisons. Si l'on donne à un tel pointeur une adresse où aucun programme machine ne se trouve, le calculateur nous quitte avec les salutations usuelles sous forme de champignons atomiques. Comme de plus ces interruptions ont lieu toutes les fractions de seconde, le 'plantage' est quasiment immédiat. Il vaut donc mieux ne pas toucher à ces pointeurs !

Prenons plutôt encore un exemple de pointeur 'inoffensif'. Ce pointeur, utilisé en BASIC, donne l'adresse d'une variable. Il s'agit de la fonction VARPTR(). Grâce à la connaissance de cette adresse, on est par exemple possible de manipuler facilement du texte.

```
10 rem ** Demo de la fonction VARPTR() **
20 a$ = ' Hello !'                    : rem définition de la
                                        chaîne de caractère a$
30 a = peek(varptr(a$)+2)             : rem détermination de
                                        son adresse
40 print a$                           : rem avant...
50 poke a,65 : print a$               : rem ... et apres !
```

La chaîne de caractères 'Hello !' est devenue ***2, sans aucune manipulation de la variable A$. Grâce à l'utilisation des pointeurs, les possibilités des instructions PEEK et POKE sont donc bien plus étendues !

Voici encore un petit programme,avec lequel vous pourrez entreprendre la 'chevauchée à travers le POKEistan sauvage'. Après la saisie de l'adresse (0 provoque la fin de l'exécution), le programme affiche son contenu et demande qu'on entre la nouvelle valeur. Si l'on ne désire pas le modifier, il suffit simplement de taper 'Return' , et la séquence d'instructions est réitérée.

```
10 rem ** utilisation de PEEK et POKE **
20 input 'Adresse ';a                    : rem saisie
30 if a=0 then end                       : rem fin si 0
40 print peek(a); ' => ';                : rem ancienne valeur
50 input x$                              : rem nouvelle valeur
60 if len(x$) = 0 then 20                : rem pas de modification
70 poke a,val(x$) : goto 20              : rem et ainsi de suite
```

Ce programme devrait être sauvegarder, car nous POKErons certainement encore souvent.

2.1.5 Les piles ou stacks

Imaginez que vous écriviez un programme machine et que vous deviez stocker un petit nombre de données quelque part. Une question se pose : où allez-vous les déposer, sans effacer quelque chose d'important et sans qu'elles soient modifiées par un autre programme ? Pour pallier ce problème, les constructeurs de microprocesseurs sont partis de l'observation suivante. Ils ont certainement observé le dessus de leur bureau et se sont dit 'J'entasse mes feuilles sur une pile, pourquoi l'ordinateur ne le pourrait-il pas ? '.

De cette idée a germé le principe du stack, qui n'est rien d'autre qu'une pile. On alloue au microprocesseur une zone mémoire particulière, dans laquelle il peut empiler et dépiler ses données. Le MC 68000 implanté dans l'ATARI ST en possède même deux, une pour le mode superviseur et l'autre pour le mode utilsateur. Ce microprocesseur a même la possibilité de créer d'autres piles pour des tâches particulières, que vous pouvez utiliser pour vos propres programmes.

Ces piles, tout comme les tas de feuilles des constructeurs imaginaires de microprocesseurs, sont des piles LIFO. LIFO veut dire Last In-First Out. Ce sont donc les données empilées en dernier qui sont les premières à être dépilées. A l'opposé, il y a les piles FIFO (First In-Last Out) qui jouent un grand rôle dans d'autres processus. Mais revenons à la pile LIFO du micro-processeur. On y empile non seulement des données, mais aussi des adresses. Cela constitue d'ailleurs l'utilisation principale d'une pile.

Quand le microprocesseur se branche à un sous-programme machine, ce qui correspond à l'instruction BASIC GOSUB, il empile la valeur du compteur ordinal (adresse que traite momentanément le microprocesseur) dans la pile. Quand arrive un RTS, qui correspond au RETURN du BASIC, il dépile cette adresse, s'y branche et continue séquentiellement. Ce processus est aussi valable pour les sous-programmes en BASIC.

De cette manière, nous pouvons facilement, à l'intérieur d'un programme, déposer provisoirement des données dont nous n'avons momentanément pas besoin. Pour un nombre de données plus grand, il faut cependant utiliser une zone mémoire propre. Si l'on ne dispose pas d'une place suffisante dans la mémoire de travail de l'ordinateur, il faut alors déposer ses données à l'extérieur du calculateur principal. A cet effet, les disquettes, que nous allons maintenant examinées, conviennent particulièrement bien.

2.2 LES DISQUETTES

Une méthode très souvent employée pour la sécurité des données est le stockage sur disquette. Le lecteur ATARI utilise des disquettes de 3,5 pouces qui sont, il est vrai, (encore) un peu chères , mais en contrepartie très pratiques et sûres. Actuellement, le format le plus répendu est le format de 5,25 pouces ; pour des installations plus grandes, on trouve également les lecteurs 8 pouces. La technique de stockage reste cependant la même pour tous les formats.

Si vous enlevez l'enveloppe d'une disquette de 3,5 pouces, vous apercevrez un disque brun foncé d'un diamètre de 3,5 pouces. Ce disque est composé de pistes constituées d'un matériau magnétique, comme une bande magnétique. Il tourne à l'intérieur du lecteur à vitesse constante. La tête de lecture se place sur le disque et lit ou écrit les informations. Avec un lecteur simple-face, seule la face inférieure de la disquette est utilisée ; la face supérieure d'une disquette simple-face n'est pas testée, de telle sorte que l'utilisation d'une disquette simple-face avec un lecteur double-face comporte le risque d'entraîner une perte de données, à cause de l'effet appelé 'Dropouts'.

La disquette est divisée en pistes et en secteurs. Chaque piste, appelée aussi Track, comprend 9 secteurs de 512 octets chacun. Le lecteur de disquettes ATARI 354 gère 80 pistes sur une disquette, ce qui donne une capacité-mémoire totale de 512*9*80 = 368640 octets. Le Floppy-Controller dans l'ATARI ST envoie, suivant le cas, le contenu d'un secteur ou d'une piste entière. L'utilisateur n'a pas besoin de savoir dans quel secteur se trouve son programme, c'est le système d'exploitation lui-même qui se charge de gérer ces informations. Elles se trouvent dans le répertoire de la disquette, où sont indiqués, outre le nom, la taille, et la date du programme, sa position et son statut (Lire/écrire ou seulement lire, voir info-fichier dans le menu).

Ce répertoire se trouve en totalité dans la mémoire. Le TOS, lors de l'appel d'un programme, y cherche les paramètres qui lui sont nécessaires. Cela comporte un risque : l'ordinateur ne remarque pas toujours que vous avez changez de disquette. Si donc vous appelez un programme de l'ancien répertoire, et bien que vous ayez changé de disquette, le système d'exploitation charge les secteurs dans lesquels devrait normalement se trouver le programme. Les données étant dans ce cas souvent entièrement fausses, cela peut conduire au 'plantage' de l'appareil.

2.2.1 Accès direct aux fichiers sur disquettes

Si vous voulez utiliser une disquette pour y stocker des informations quelconques (par ex. du texte), vous devez d'abord ouvrir un fichier. A cet effet, le BASIC possède l'instruction OPEN. Voici un exemple de programme utilisant cette instruction :

```
5  input 'nom du fichier';f$        : rem par ex. TEST.TXT
10 input '(1) saisie / (2) affichage / (3) fin';a
20 on a goto 50,100
30 end
40 rem ** saisie du texte **
50 print 'saisie : ('x' = break)'
60 open 'O',#1,f$,128                : rem O comme Output
70 input a$
80 print #1,a$ : if a$='x' then close #1 : goto 10
90 goto 70
95 rem ** affichage du texte **
```

```
100 print 'affichage :'
110 open 'I',#1,f$,128                    : rem I comme Input
120 input #1,a$ : if a$='x' then close #1 : goto 10
130 print a$ : goto 120
```

Ce programme montre l'emploi des instructions OPEN, INPUT# et PRINT#. Le sens de cheminement des données (vers ou de la disquette) doit être défini dans l'instruction OPEN. Avec l'ordre PRINT#, vous pouvez envoyer sur disquette n'importe quels textes ou valeurs numériques, et les relire avec INPUT#.

2.2.2 Sauvegarde d'une image graphique

Une autre manière de stocker des données est d'utiliser les instructions BLOAD et BSAVE. Grâce à celles-ci, on peut directement charger/sauvegarder des zones mémoires à partir de la disquette. Vous pouvez ainsi, par exemple, charger un programme machine, que vous avez écrit vous-même, à une adresse bien définie, et l'appeler grâce à un CALL. Vous pouvez de la même façon sauvegarder la mémoire écran, si vous avez créer une image graphique, et cela de la manière suivante :

```
110 rem ** sauvegarde de la mémoire écran **
120 defdbl a : a=1102
130 b = peek(a)
140 bsave 'image.dat', b, 10000
150 return
200 rem ** chargement de l'image **
202 defdbl a : a=1102
204 b=peek(a)
206 bload 'image.dat', b
210 return
```

Les deux parties du programme peuvent être appelées comme des sous-programmes par GOSUB 100 (ou GOSUB 200). Le nom de fichier 'IMAGE.DAT' a été choisi de manière tout à fait arbitraire et peut être modifié. Cependant il doit comporter au plus 8 caractères et un type de 3 caractères (ici .DAT), le point ne comptant pas. Les caractères qui dépassent sont ignorés. De plus, il faut que le premier caractère du nom soit une lettre (A-Z).

3. LA REPRESENTATION INTERNE DES NOMBRES ET LEUR MANIPULATION

En travaillant sur ordinateur, en particulier avec le langage machine et les instructions BASIC apparentées PEEK et POKE, on rencontre toujours différentes représentations des nombres. C'est pourquoi nous allons entreprendre une petite excursion à travers le système numérique. Si ces notions vous sont familières, vous pouvez tranquillement sauter ces pages, car vous n'apprendrez certainement rien de nouveau.

Toutes les adresses indiquées jusqu'ici dans les tables sont écrites dans le système hexadécimal. Normalement, elles sont précédées par un '$' ou un 'H'. Il s'agit d'un système numérique, qui se distingue du système décimal bien connu par le fait qu'il apparait, outre les chiffres de 0 à 9, les lettres A à F. La raison, pour laquelle on préfère cette notation quelque peu inhabituelle, réside premièrement dans le nombre plus petit de caractères qui sont nécessaires pour écrire ces valeurs élevées ($FC 0000 est égal à 16.515.072 en décimal).

D'un autre coté, et ceci constitue son avantage décisif, le système hexadécimal décrit beaucoup plus clairement la configuration de la mémoire d'un calculateur digital. Ainsi, un octet peut prendre une valeur comprise entre 0 et 255, donc 256 états différents. Si maintenant, on ne prend plus 10 pour base numérique comme dans le système décimal (déci=dix), mais 16, on peut, avec deux chiffres, atteindre exactement la valeur 255,c'est à dire $FF. L'ensemble des caractères passe de 0..9 à 0..F, où A correspond à 10, B à 11,etc...

Cependant, le microprocesseur qui doit travailler avec ces nombres n'utilise ni le système décimal ni le système héxadimal. Pour lui, il n'y a que deux états possibles, 5 Volt ou 0 Volt. Cette représentation de l'information est la seule que l'ordinateur est capable de comprendre et de traiter directement. C'est le domaine machine.

Toutes les données et adresses sont composées d'une suite de zéros et de uns. Cette technique de codage au moyen de deux états est appelée digitale (di=deux). Prenons par exemple le nombre 100. L'octet correspondant contient en réalité la suite '01100100', où chaque représente un potentiel bivalent, 0V ou 5V. Le bit de droite a donc la valeur 2^0 (1), le suivant 2^1(2), puis 2^2(4), etc... Ces valeurs sont additionnées, quand le bit correspondant est à 1. Cela donne dans notre exemple :

2^2 + 2^5 + 2^6=4 + 32 + 64 = 100

Cette méthode semble quelque peu compliquée quand on pense aux 24 bits constituant chaque adresse de l'ATARI. Pourtant elle a beaucoup d'avantages quand on connait les différents systèmes numériques. C'est pourquoi il est bon de savoir convertir les nombres d'un système numérique dans un autre.

3.1 CONVERSION DES SYSTEMES NUMERIQUES

Cette conversion est, il faut bien l'avouer, fastidieuse, si on veut l'effectuer à la main. Mais au fait, pourquoi possède-t-on un ordinateur ? Ecrivons donc un programme BASIC qui le fera à notre place ! Voici des exemples de programmes, qui effectuent différentes conversions. Malheureusement, j'ai dû avoir recours à quelques artifices de calcul, car la précision de calcul de la version BASIC dont je disposais laissait quelque peu à désirer. Par ailleurs, il faut encore mentionner l'instruction HEX$, qui n'est cependant utilisable qu'avec des valeurs constantes, par ex. PRINT HEX$(100) ou des variables entières, par ex. PRINT HEX$(x%), et qui de ce fait ne donne que des résultats inférieurs à $FFFF.

```
10  defdbl e,d                      : rem déclaration de variables
20  defint n
30  z$= '0123456789ABCDEF'
40  input 'nombre decimal ';d       : rem  Decimal => Hexadecimal
50  for i=5 to 0 step -1 : e=16^i
60  n=.1+d/e : d= d-n*e
70  print mid$(z$,n+1,1);
80  next i : ? : ? : end
```

```
90 defdbl e,d                          : rem déclaration de variables
100 defint n
110 z$='0123456789ABCDEF'
120 input 'nombre hexadecimal ';h$  : rem Hexadecimal => Decimal
130 d=0 : for i=1 to len(h$)
140 e=16^(len(h$)-i)
150 d=d + e*(instr(1,z$,mid$(h$,i,1))-1)
160 next i : print d : end

170 input 'nombre decimal ';d          : rem Decimal => Binaire
180 print d;' => '; : for i = 15 to 0 step -1
190 if (d and 2^i) then ? '1'; else print '0';
200 next i : print : end

210 input 'nombre binaire ';b$         : rem Binaire => Decimal
220 print b$;' = '; : d = 0
230 for i=1 to len(b$) : e = len(b$)-i
240 d=d-(mid$(b$,i,1)='1')*2^e
250 next i : print d : end
```

3.2 MANIPULATION DE BIT (Par ex. pour la gestion du joystick)

C'est précisément la représentation binaire des nombres qui joue un grand rôle dans la manipulation de la mémoire. En effet, certaines fonctions dépendent souvent d'un bit précis, comme par exemple la détermination de la position du joystick. Le nombre obtenu en interrogeant l'état du joystick donne dans quatre bits les états des quatres interrupteurs, qui sont actionnés en bougeant la poignée. Les bits 0 et 1 indiquent la position verticale du joystick, les bits 2 et 3 la position horizontale. Voici un petit exemple de programme illustrant ce propos :

```
10 rem ** gestion du joystick **
20 out 4,22                    : rem directive destinée au clavier
25 DEFSEG = 1
30 js = peek(3592)             : rem lire le résultat
40 print js,                   : rem afficher la valeur totale
50 Y=Y + (js and 2)/2 -(js and 1)
60 X=X + (js and 8)/8 -(js and 4)/4
70 print X,Y                   : rem afficher les coordonnées
80 goto 20                     : rem boucle sans fin
```

Ce programme interroge le joystick 1 et modifie une position donnée par X et Y (par ex. les coordonnées d'un curseur) suivant les positions du joystick. Ici les bits respectifs sont testés grâce à l'instruction AND. Cette relation AND est proche parente des fonctions BASIC OR, NOT et XOR. Ces relations sont appelées binaires, car elles comparent les deux valeurs bit par bit. Le résultat de cette comparaison va vous être expliqué dans le chapître suivant.

3.3 LES OPERATIONS LOGIQUES

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0   Le bit-résultat n'est mis à 1,
1 AND 1 = 1   que si les deux bits testés sont à 1.

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1    Le bit-résulat est toujours mis à 1,
1 OR 1 = 1    dès lors qu'un des bits testés est à 1.

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1   Le bit-résultat est mis à 1,
1 XOR 1 = 1   si les bits testés sont différents
```

Enfin, l'opération NOT, qui inverse la valeur d'un bit :

```
NOT 0 = 1
NOT 1 = 0
```

Le programme de l'exemple ci-dessus teste la position du joystick en 'masquant' la valeur JS grâce à un AND. Si cette valeur est 3 et que l'on veut tester le premier bit (bit numéro 0), l'opération JS AND 1 donne le résultat suivant :

```
JS          = 3, en binaire :  0011
octet-test  = 1,               0001
                               ----
résultat    3 AND 1            0001
```

Le résultat est différent de zéro que si le bit testé est égal à 1. De cette manière, l'état du joystick est entièrement connu avec quatre de ces comparaisons.

Une autre utilisation des opérateurs logiques est la manipulation d'un bit. Si l'on a affaire à un octet, dont on veut effacer un bit précis, on utilise aussi l'opérateur logique AND.

Exemple :

```
L'octet vaut      00110011 Le bit 4 doit être éffacé, donc
          AND     11101111
                  --------
                  00100011 le bit est effacé !
```

Cette méthode est appelée masquage, le masque étant ici l'octet 11101111. Une méthode analogue est employée pour forcer un bit à 1. On utilise alors l'opérateur OR.

Exemple :

```
L'octet vaut      00110011 Le bit 3 doit être forcé à 1, donc
           OR     00001000
                  --------
                  00111011 le bit est à 1 !
```

L'opérateur XOR offre une autre possibilité intéressante. Grâce à lui, on peut inverser des bits, c'est à dire qu'un bit à 1 est effacé et inversement. Si la même opération est répétée plusieurs fois, le bit prend alternativement la valeur 0 et 1.

Exemple :

```
L'octet vaut      00110011 Le bit 0 doit être inversé
          XOR     00000001
                  --------
                  00110010 encore une fois
          XOR     00000001
                  --------
                  00110011 on retrouve la valeur initiale!
```

Enfin, il faut encore parler de l'opérateur NOT, qui possède une certaine similitude avec XOR. NOT inverse tous les bits d'un octet. Le résulat est donc égal à 255 (décimal) moins la valeur initiale.

Exemple :

L'octet vaut 00110011 = 51

NOT 00110011 donne
11001100 = 204 (204 + 51 = 255)

Ces opérateurs logiques sont utilisables sous BASIC avec les instructions correspondantes. En langage machine, cependant, existent encore d'autres opérateurs de manipulation de bits. On peut décaler un mot vers la droite ou vers la gauche, ou effectuer une rotation sur tous ses bits. L'effet d'une telle opération est la multiplication (ou la division) du mot par 2.

Un exemple :

L'octet vaut	00110011 = 51
décalage à gauche	01100110 = 102 = 51*2
décalage à droite	00011001 = 25, c'est à dire la partie entière de 51/2

Lors d'un décalage, l'octet est comblé avec des 0. En résumé, cela signifie que n décalages vers la gauche correspondent à la multiplication par 2^n, et n décalages vers la droite à la division par 2^n.

Un autre problème lié à l'arithmérique binaire est la représentation binaire des nombres négatifs. On utilise toujours le bit de plus fort poids comme signe. Si ce bit est égal à 1, il s'agit dans certains cas, d'un nombre négatif ; cependant, la prise en compte du signe n'est pas toujours nécessaire, car certains nombres, comme les adresses par exemple, prennent toujours une valeur positive. En utilisant l'instruction PEEK, on remarque que toutes les valeurs supérieures à 32767 sont représentées négativement. Après un POKE x,65535 on obtient, par un PEEK(x) la valeur -1 ; cette valeur négative est obtenue, pour les nombres supérieurs à 32767, en soustrayant la valeur initiale à 65536 (ici -1 = -(65536-65535)).

Cette relation entre un nombre et sa représentation négative est appelée complémentation à 2. L'instruction NEG du langage machine MC 68000 réalise cette complémentation, en soustrayant la valeur initiale de 0.

Une autre représentation des nombres, à l'intérieur de l'ordinateur, est le codage BCD. BCD veut dire Binary Coded Decimal. Ce code est utilisé pour représenter les nombres décimaux. Ici, on divise le mot binaire en blocs de 4 bits, chaque bloc contenant un chiffre (0 à 9). Un mot de 16 bits n'est donc pas utilisé pleinement, puisque la représentation BCD peut seulement atteindre la valeur 9999, alors qu'on atteint 65535 avec un codage normal.

L'avantage de ce code est de pouvoir représenter et coder facilement les nombres décimaux. L'évaluation d'un nombre décimal codé en BCD est aussi simple que l'évaluation de la valeur héxadécimale d'un nombre binaire normal. Il suffit de transformer séparément chaque bloc en un chiffre, et l'on obtient directement la valeur codée. Vous trouverez un exemple de codage BCD dans le programme d'évaluation de l'heure dans le chapitre concernant le clavier. L'heure, tout comme la date, est en effet codée en BCD avant d'être mémorisée.

4. LE SYSTEME D'EXPLOITATION

Un ordinateur est, au moment de la mise en marche, véritablement bête. Même s'il a énormément de possibilités, il ne sait pas comment les utiliser. Un programme lui indiquant quoi faire est donc indispensable. Un tel programme, qui dirige la totalité de l'ordinateur, est appelé système d'exploitation. Certains de ces programmmes systèmes ont déjà fait leurs preuves, comme par exemple CP/M qui travaille avec le microprocesseur Z80 A, ou MS-DOS, qui peut travailler avec des microprocesseurs appelés des faux 16 bits comme le 8088 de l'IBM-PC. En liaison avec le MC 68000, il y a le CP/M 68K, qui devait initialement être implanté dans l'ATARI ST. Mais ce système d'exploitation a certains inconvénients, comme par exemple sa vitesse qui laisse à désirer. Par ailleurs, il n'est pas en mesure de gérer les vastes possibilités du ST. Ainsi, la société de software Digital Research, qui a également crée CP/M, a développé le système d'exploitation GEMDOS spécialement conçu pour la série ST. Ce système d'exploitation est également connu sous le nom de TOS.

4.1 LE TRAMIEL-OPERATING-SYSTEM

Cette dénomination du GEMDOS est due au 'père' de l'ATARI ST. L'une des activités du TOS est donc de gérer les unités de disquettes et les autres périphériques, d'une manière similaire à CP/M. Cependant, le TOS a encore bien d'autres tâches. Pour répondre aux multiples besoins de l'ATARI ST, l'utilisation de CP/M aurait demandé trop de modifications. CP/M 68K, c'est ainsi que s'appelle la version de ce système d'exploitation pour le processeur MC 68000, a par ailleurs l'inconvénient de ne pas travailler très vite. C'est pourquoi ATARI a décidé de développer un système d'exploitation entièrement neuf pour l'ATARI ST. Bien que quelques idées du CP/M 68K aient été maintenues, le GEMDOS a été nettement amélioré et et possède de nombreuses nouveautés. La structure hiérarchique du repertoire, connue dans MS/DOS (un système d'exploitation IBM), est également réalisée dans le ST. Grâce à cette structure, le possesseur d'un ATARI ST a la possibilité de créer des sous-directories.

4.1.1 Le BIOS

Une partie importante du TOS est le BIOS (Basic Input Output System), qui réalise la liaison entre le software et le hardware. Le BIOS peut être programmé de manière très simple, ce qui constitue un véritable soulagement pour le programmeur. Ainsi par exemple, ce dernier n'a pas à se faire de soucis au sujet de l'enchaînement des signaux du port parallèle, mais donne simplement au BIOS un ordre d'entrée ou de sortie. Cependant, la programmation du BIOS n'est réellement intéressante que pour celui qui connait le langage C ou le langage machine. Les utilisateurs du BASIC ont moins de possibilitées, mais également plus de facilités : il suffit d'entrer PRINT et l'interprèteur BASIC fait le reste.

Mais le BIOS n'est pas suffisant pour un ordinateur du calibre du ST. C'est pourquoi le BIOS a été amélioré en ajoutant quelques instructions supplémentaires. Parmi celles-ci, il y a les opérations sur l'écran, sur les disquettes, sur le port MIDI, ainsi que la fonction Hardcopy implantée. La liste des 52 instructions du BIOS et de l'extended BIOS (XBIOS) est cependant inutile, car les instructions spéciales de lecture, de transmission et d'envoi de paramètres ne sont utilisables que par des programmeurs proches de la machine (voir paragraphes 6.3 et 6.4). Les fans de BASIC évite ainsi de compulser ces tables. Il y a bien sûr des fonctions du BIOS qui ne sont pas accessible sous BASIC. Il n'est par exemple pas possible de configurer (uniquement à partir du DESKTOP) l'interface de l'imprimante ou du modem. On peut s'accomoder de ces inconvénients, quand on pense au confort d'un langage évolué comme le BASIC. Mais une porte dérobée est laissée ouverte par ce langage : les instructions PEEK et POKE. Grâce à elles, on peut modifier certains paramètres du système, dont l'accès est normalement réservé au TOS.

La version du TOS étudiée ici est chargée à partir de la disquette quand l'ordinateur est mis en route. Il occupe la zone mémoire $500 à $32000 et contient le logiciel graphique GEM et les paramètres nécessaires à la réalisation des menus et des fenêtres. Cette gigantesque zone mémoire n'est pas uniquement occupée par les programmes, on y trouve également quelques paramètres sur l'état du système au démarrage. Quand le TOS est complètement chargé, le BIOS entre en action (à partir de $5000) et initialise l'ensemble de l'ordinateur.

Puis il cherche sur la disquette les programmes de type .ACC et les charge en mémoire. Ces programmes appelés accessory-programs sont alors accessibles en permanence et peuvent être appelés par l'utilisateur à partir du menu DESK. Les deux fichiers ACC, disponibles sur la disquette fournie, contiennent le tableau de contrôle, la configurations des interfaces et le Terminal-Emulator. Si vous n'avez pas besoin de ces programmes, vous pouvez économiser la place mémoire correspondante, en changeant leur nom (par ex. DESK1.AC). Vous gagnerez ainsi environ 20 Koctets pour votre programme BASIC.

4.1.2 Les variables systèmes

Le BIOS initialise encore une autre zone mémoire. Il s'agit d'une table de paramètres systèmes, qui peut être modifiée en partie. Cette zone comprend également beaucoup de pointeurs, qui pointent sur des programmes machines d'interruption. On ne devrait en aucun cas toucher à ces pointeurs, car leur modification entraînerait très certainement le plantage du système (et à nouveau l'apparition des champignons atomiques à l'écran...). Cette table se trouve entre les adresses $400 et $4FF (1024-1279) et contient les variables systèmes suivantes :

$400 - $41F	adresses de saut (pointeurs).
$420	contient un 'nombre magique', qui indique si un redémarrage 'à froid' a réussi.
$424	configuration mémoire(=4 avec 512 Koctets, 5 avec 1 Moctet).
$426	encore un nombre magique, qui provoque, lors d'un Reset, un redémarrage à froid.
$42A	pointeur de la routine de redémarrage à froid.
$42E	pointeur de fin de RAM ($80000 avec 512 Koctets, $F0000 avec 1 Moctet).
$432	pointeur du début de la mémoire de travail.
$436	pointeur de fin de la mémoire de travail.
$43A	encore à nouveau un nombre magique, qui indique un redémarrage à froid réussi.
$43E	Flag DMA , il doit être <>0 !
$440	vitesse de la disquette (3).
$442	System-Timer en millième de seconde (20 pour 50 Hz)

$444	Flag indiquant le mode d'écriture sur disquette. Si <> 0, alors chaque processus d'écriture est testé par une lecture. Si 0, ce n'est pas le cas (écriture plus rapide).
$446	numéro du lecteur, d'où le système est chargé.
$448	Flag PAL/NTSC (<>0 : PAL ; =0 : NTSC).
$44A	résolution graphique initiale.
$44C	résolution graphique actuelle (0-2).
$44E	pointeur de début de la mémoire écran.
$452	Flag VBI. Devrait être égal à 1 (VBI = Vertical Blank Interrupt).
$454	numéro d'une routine VBI (8).
$456	pointeur de la table des pointeurs VBI.
$45A	pointeur de la nouvelle table des couleurs (si nouvelle il y a).
$45E	pointeur de la nouvelle mémoire écran (si nouvelle il y a).
$462	compteur VBI.
$46A	compteur des routines VBI exécutées.
$46E	pointeur de la routine appelée lors d'un changement de moniteur (redémarrage à froid).
$472-$481	paramètres liés au disque dur (0 s'il n'y en a pas)
$482	si <>0, COMMAND.PRG a été chargé.
$484	état du clavier (bit 0: horloge on/off; bit 1: Key-repeat on/off; bit 2: beep des touches on/off).
$48E	limites de la mémoire de travail (ne pas modifier!)
$4A2	pointeur du tableau des registres du BIOS.
$4A6	nombre de lecteurs de disquettes connectés.
$4AE	pointeur de la zone mémoire indiquant l'état du microprocesseur.
$4B4	pointeur du buffer contenant les informations d'un secteur.
$4B8	pointeur sur le buffer contenant le répertoire.
$4BC	compteur 200 Hz.
$4C4	=3, si lecteurs de disquettes branchés.
$4C6	pointeur du buffer-Disk de 1 Koctets.
$4EE	flag Hardcopy (si 0, alors impression de la mémoire écran).
$4F2	pointe sur l'adresse de début du système d'exploitation.

$4F6	pointe sur le segment graphique ou le segment texte
$4FA	pointe sur la fin du système d'exploitation.
$4FE	pointe sur le segment texte AES.

4.1.3 L'écran TOS

L'environnement utilisateur du Desktop du GEM n'est pas le seul dont dispose l'ATARI ST. Il en existe encore un autre, qui n'est cependant utilisable qu'en mode texte. Sur cette 'feuille de papier' travaillent tous les programmes, qui ont pour type '.TOS'. L'écran TOS est aussi utilisé avec le VT52-Emulator qui est une sous-option du point de menu DESK. Dans cet éditeur, tous les mouvements du curseur sont commandés par des caractères de contrôle, qui commence tous par la touche Escape. Dans la table suivante, vous trouverez les différentes séquences de touches et leur effet. Esc est une abréviation pour Escape.

Esc A	curseur vers le haut
Esc B	curseur vers le bas
Esc C	curseur à droite
Esc D	curseur à gauche
Esc E	efface l'écran et place le curseur en haut à gauche
Esc H	curseur dans le coin supérieur à gauche
Esc I	curseur vers le haut, scrolling si nécessaire
Esc J	efface l'écran à partir de la position du curseur
Esc K	efface la ligne à partir de la position du curseur
Esc L	insère une ligne
Esc M	efface la ligne
Esc Y (y-32)(x-32)	amène le curseur à la position X/Y
Esc b couleur	choix de la couleur de l'encre (0-16 pour une image couleur)
Esc c couleur	couleur du fond (0-16)
Esc d	efface l'écran jusqu'à la position du curseur
Esc e	active le curseur
Esc f	désactive le curseur
Esc j	mémorise la position du curseur
Esc k	place le curseur à la position mémorisée

Esc l	efface la ligne
Esc o	efface la ligne jusqu'au curseur
Esc p	active l'inversion video
Esc q	désactive l'inversion vidéo
Esc v	dépassement autorisé
Esc w	dépassement non autorisé (le curseur s'arrête sur la colonne de droite)

Ces fonctions sont, comme je l'ai dit, valables dans le VT52-Emulator. Mais on peut aussi les utiliser dans un programme BASIC. Il est ainsi possible d'effacer le cadre de la fenêtre OUTPUT et d'utiliser tout l'écran pour le programme. Attention cependant : une ligne du menu une fois effacée reste effacée, bien qu'elle soit utilisable. Il faut donc la réécrire, après être revenu dans l'éditeur BASIC normal.

Le programme suivant met ces notions en pratique :

```
10 rem ** Demo - Editeur TOS **
20 x = inp(2)              : rem recherche de la touche enfoncée
30 if x=187 then end       : rem F1 provoque la fin de l'exécution
40 out 2,x                 : rem envoi à l'écran TOS
50 goto 20                 : rem boucle sans fin
```

Cette astuce peut également être utilisée dans un programme BASIC pour obtenir deux curseurs indépendants. Pour pouvoir déplacer le curseur TOS à partir d'un programme BASIC et pour afficher un message à cet endroit, on peut utiliser ce programme :

```
10 rem ** affichage à la position du curseur TOS **
20 esc$=chr$(27)                 : rem définir le caractère escape
30 message$=esc$+'B'+esc$+'b2': rem curseur vers le bas et couleur
40 message$=message$+'Hello !': rem texte
50 for i=1 to len(message$)      : rem boucle d'envoi
60 out 2,asc(mid$(message$,i,1))  : rem envoyer le caractère
70 next i                        : rem jusqu'à 'épuisement du stock'
```

La variable MESSAGE$ peut aussi être définie d'une autre manière. Ici, la boucle d'envoi transmet au TOS le texte en entier incluant les caractères de contrôles. Tous les caractères de contrôle indiqués plus haut peuvent être employés. Du reste, si la fréquence de clignotement du curseur ne vous convient pas, vous pouvez la modifier avec POKE 9572,n.

A ce propos, il existe un autre moyen pour utiliser la notion de durée dans un programme. Le nombre contenu à l'adresse 9572 indique en effet la valeur qui est chargée à l'adresse suivante après chaque modification de l'état du curseur. La valeur contenue à l'adresse 9573 est alors décrémentée jusqu'à zéro. Ce décomptage a lieu automatiquement, de telle sorte qu'au bout d'une période de temps bien précise, la valeur 0 est atteinte et le curseur change d'état. On peut utiliser ce compteur dans un programme, en affectant une durée à l'adresse 9572, et en attendant à l'intérieur d'une boucle jusqu'à ce que le contenu de l'adresse 9573 passe à 0. Dans cette boucle, on peut également attendre un autre événement. Ainsi un programme peut attendre une saisie en laissant seulement un certain temps à l'utilisateur (test de rapidité).

Comme nous en sommes aux temps d'attente et aux boucles d'attente, il faut encore mentionner que l'on peut aussi attendre un temps déterminé grâce à l'instruction SOUND. Ce faisant, il suffit de rendre le son imperceptible à l'oreille et de prendre la durée du son comme temps d'attente. Cette durée est exprimée en 1/50 de secondes. L'instruction SOUND 1,0,1,1,50 provoque par exemple un arrêt du programme de 1 seconde.

4.2 LE GEM

Le GEM, Graphics Environnement Manager, a été développé par Digital Research pour fournir au ST une nouvelle interface utilisateur. Cela rappelle l'environnement du Mackintosh de APPLE, qui créa pour la première fois un ordinateur orienté vers l'art graphique. La méthode traditionnelle, qui consiste à envoyer des instructions à partir du clavier, est remplacée par le déplacement de symboles à l'écran et l'utilisation de menus. Un des avantage est la grande facilité d'apprentissage du maniement de l'ordinateur - la plupart des fonctions s'expliquent par elles-mêmes. De plus, on évite de feuilleter dans des manuels d'utilisation pour trouver la signification de telle ou telle instruction.

Il serait naturellement intéressant de pouvoir exploiter les possibilités du GEM dans des programmes BASIC. Cela est possible car toutes les fonctions du GEM, qui sont nécessaires à la réalisation de graphismes et à la gestion des menus, sont accessibles. Mais avant de passer à la programmation, il faut se faire une idée plus précise de la façon dont travaille le GEM et comment il est organisé.

Le GEM se compose en fait de deux programmes distincts : le VDI (Virtual Device Interface) et l'AES (Application Environment System). Bien que tous les deux s'appliquent à des domaines d'utilisation semblables, ils ont des tâches différentes.

Le VDI s'occupe de la réalisation des graphiques. Il se compose du GDOS (Graphic Device Operating Systeme), qui contient les fonctions graphiques, et le GIOS (Graphic Input/Output System). Ce dernier gère les entrée/sorties graphiques. Il possède, pour chaque périphérique possible, un circuit spécifique. Cela permet l'indépendance totale entre les divers types de périphérique, qu'il s'agisse du moniteur, de l'imprimante ou des lecteurs de disquette, de telle sorte que les fonctions graphiques GDOS sont elles aussi réalisées indépendamment du microprocesseur. C'est pourquoi il est possible de faire tourner un programme qui a été écrit en GEM sur un autre ordinateur.

L'AES, quant à lui, s'occupe de la conduite de l'utilisateur, c'est à dire qu'il gére les menus déroulants et les icônes (les symboles pour les unités de disquettes, la corbeille à papier etc.) et toute la technique des fenêtres, en liaison avec la souris. L'AES est donc un système d'exploitation puissant.

4.2.1 La programmation du GEM en BASIC

Les deux parties du GEM sont écrites en langage C et en assembleur. Dans le kit de développement qu'ATARI a développé pour les programmeurs et qui malheureusement n'est pas fourni avec l'ensemble, on trouve les bibliothèques des paramètres GEM. Le programmeur C peut, grâce à un appel relativement simple des routines AES et VDI, concevoir et utiliser ses propres graphiques et menus. Mais les auteurs de l'interprèteur BASIC ont heureusement rendu le GEM accessible à leur langage.

L'ensemble des instructions BASIC comprend en effet les instructions GEMSYS et VDISYS. VDISYS est, comme on peut le penser aisément, un appel du VDI ; par contre, GEMSYS fait intervenir uniquement l'AES. Grâce à ces deux instructions, il est donc possible d'appeller directement le GEM à partir d'un programme BASIC. Cependant, un appel seul ne suffit pas. L'AES et le VDI ont en effet besoin de savoir, quoi faire et avec quels paramètres.

Les créateurs du BASIC de l'ATARI ont également penser à ce problème. Il existe ici, comme en C, des tableaux dans lesquels le programme doit transmettre les paramètres. Il s'agit des constantes INTIN, INTOUT, PTSIN, PTSOUT et CONTRL. On peut interroger leur contenu avec PRINT INTIN, INTOUT, ... Les valeurs obtenues sont les adresses mémoires où commencent les tableaux des paramètres. Pour affecter par exemple une valeur de fonction au deuxième élément de INTIN, il faut ajouter 2 à l'adresse de INTIN, car les tableaux contiennent des mots de 16 bits. L'élément INTIN(3) a donc l'adresse INTIN+6.

Ces tableaux, appelés aussi champs ou arrays, ont chacun un rôle particulier. Le tableau INTIN sert à transmettre des paramètres d'entrée, alors que dans INTOUT sont stockées des valeurs de sortie. PTSIN et PTSOUT sont en général utilisés pour l'envoi de coordonnées, par exemple pour une sortie graphique. Le tableau le plus important pour la gestion des fonctions est le tableau CONTRL, dans lequel des paramètres sont à la fois reçus et émis. Il contient les éléments suivants :

CONTRL(0)	code instruction de la fonction (numéro de la fonction)
CONTRL(1)	nombre d'éléments dans le tableau PTSIN
CONTRL(2)	nombre d'éléments dans le tableau PTSOUT
CONTRL(3)	nombre d'éléments dans le tableau INTIN
CONTRL(4)	nombre d'éléments dans le tableau INTOUT
CONTRL(5)	identificateur de fonction (ID) pour les sous-fonctions
CONTRL(6)	identificateur de périphérique (Handle)
CONTRL(7-n)	Valeur dépendant de la fonction

Les éléments 2, 4 et 6-n sont des paramètres de sortie, que renvoie la fonction. Les autres doivent en fait être donnés à chaque appel de fonction. L'élément CONTRL(6) est un nombre lié au périphérique utilisé.

Mais comme l'écran est de toute façon allumé et représente donc le périphérique utilisé, cet identificateur n'a pas besoin d'être transmis. Toutes les fonctions sont de cette façon envoyées à l'écran.

4.2.2 Gestion de la souris

Nous allons laisser de côté la théorie et écrire un petit programme qui détermine à la fois la position de la souris à l'écran et l'état des deux touches de la souris. Cela fait partie des compétences du VDI et constitue une application de l'appel du VDISYS. Mais passons maintenant au programme :

```
10  rem ** Position de la souris **
20  poke contrl,124              : rem code de l'instruction
30  poke contrl+2,0              : rem nombre de paramètres
70  vdisys 0                     : rem et exécution
80  x = peek(ptsout)             : rem position X
90  y = peek(ptsout+2)           : rem position Y
100 touche = peek(intout)        : rem touche enfoncée
110 print x,y,touche
120 goto 20
```

A la ligne 20, on transmet tout d'abord le code instruction dans CONTRL(0). Ceci doit d'ailleurs être fait à chaque appel du GEM. Puis, on communique le nombre de paramètres transmis, qui est, il est vrai, égal à zéro dans cet exemple, aucune valeur d'entrée n'étant nécessaire à cette fonction. Enfin, il y a l'appel VDI. Le zéro après l'instruction VDISYS est arbitraire, une valeur doit cependant être entrée ('Dummy').

De son coté, le VDI donne maintenant des valeurs en retour. Ces valeurs de sortie se trouvent toujours dans INTOUT ou PTSOUT, ces champs étant uniquement prévus pour des sorties. Dans de rares cas, on obtient aussi des valeurs dans la table CONTRL, mais les deux champs OUT sont plus importants.

Dans l'exemple ci-dessus, PTSOUT(0) et PTSOUT(1) contiennent les positions X et Y du curseur de la souris. Le nombre contenu dans INTOUT(0) est 0 si aucune touche de la souris n'est enfoncée. Ces données peuvent ensuite être utilisées dans la suite du programme. Les lignes 110 et 120 servent uniquement de vérification et peuvent être supprimées.

Comme nous parlons justement du curseur de la souris - ce symbole ne vous ennuie-t-il pas à force ? Voulez-vous peut-être déplacer un petit bonhomme sur l'écran ? Vous devez alors le communiquer au VDI. J'ai également préparé un programme qui le réalise. Ce programme transmet cette fois-ci un grand nombre de paramètres. Voici donc ce programme.

4.2.3 Modification de la forme de la souris

```
10  rem ** modification de la forme de la souris - demo VDI **
20  poke contrl,111              : rem code instruction
30  poke contrl+6,37             : rem nombre de paramètres
40  poke intin,3                 : rem point d'action X
50  poke intin+2,0               : rem point d'action Y
60  poke intin+6,0               : rem valeur de la couleur du masque
70  poke intin+8,1               : rem valeur de la couleur du curseur
80  for i=0 to 15 : read x$      : rem masque/curseur
82  x=0 : for j=1 to 16          : rem transformation
84  x=x-(mid$(x$,j,1)<>' ')*2^(16-j)
86  next j
90  poke intin+10+i*2,x          : rem mettre en place le masque
92  x=0 : for j=1 to 16          : rem transformation
94  x=x-(mid$(x$,j,1)='*') * 2^(16-j)
96  next j
98  poke intin+42+i*2,x          : rem mettre en place le curseur
100 next i
110 vdisys 0                     : rem et exécution
120 end
130 rem +++ données masque/curseur +++
```

```
140 data '                '
150 data '     ***        '
160 data ' **  *****      '
170 data '  **  ***        '
180 data '   **  *         '
190 data '    ********     '
200 data '     ***** **    '
210 data '     *****  *    '
220 data '     ***** **    '
230 data '     ** **       '
240 data '     ** **       '
250 data '     ** **       '
260 data '     ** **       '
270 data '   **** ****     '
280 data '                '
290 data '                '
```

Ce programme nécessite, il est vrai, quelques explications. Dans la ligne 20, on transmet comme précédemment le code instruction de la fonction VDI choisie. Puis suit la déclaration des paramètres qui sont transmis lors de cet appel. Cette fonction, dont le nom est du reste Vsc_form, lit 37 paramètres à la suite, qui sont affectés au tableau INTIN. La signification de chaque paramètre vous est déjà connue. Dans INTIN(0) et INTIN(1) se trouvent les coordonnées X et Y du point d'action du curseur de la souris qui indique le symbole qui, par exemple, active les menus déroulants. Dans cet exemple, ce point a pour coordonnées 0 et 3, c'est à dire la 'main' droite.

Les deux valeurs suivantes affectées à INTIN(3) et INTIN(4) indiquent les valeurs des couleurs du curseur. On se demande maintenant pourquoi deux couleurs sont nécessaires. La raison de cela est simple à comprendre : supposons que le curseur soit noir et soit déplacé sur une surface également noire. Il deviendrait alors invisible, de telle sorte qu'il serait difficile à localiser de façon exacte. Le masque trouve ici son utilité. Ce masque se trouve quasiment sous le curseur et est un peu plus grand que celui-ci. La valeur de sa couleur est dans l'exemple ci-dessus, comme d'ailleurs dans l'environnement Desktop normal, égale à zéro, c'est à dire qu'il apparait blanc.

L'effet est alors le suivant :

Quand on passe sur une surface noire, on ne voit effectivement pas le curseur, mais seulement son pourtour.
Et ce pourtour n'est rien d'autre que le masque.

Le programme transmet, à partir de la ligne 80, les données déterminant les formes du curseur et du masque. On utilise ici une nouvelle fois le principe de la conversion Binaire-Decimal, qui a déjà été traité dans le chapitre 2.1. Dans notre cas, la routine de conversion a été légèrement modifiée. Pour les données du motif du curseur, qui constituent une représentation binaire de ce dernier, l'étoile * est reconnu comme un 1. Par contre, le masque est représenté par les points, car il est plus grand que le curseur lui-même. Ici, c'est le caractère blanc qui correspond à la valeur logique 0, tous les autres caractères sont considérés comme des 1 par cette routine située entre les lignes 82 et 90. Grâce à cette astuce, on économise la création de deux tables différentes, une pour le curseur et une pour le masque, bien qu'en réalité deux tables soient nécessaires.

Le programme de conversion de la ligne 80 à 100 établit à partir d'une table combinée deux tables séparées et les transmet directement dans le champ VDI. Lors de l'appel du VDI par l'instruction VDISYS, les données du motif du masque se trouvent ainsi disponibles dans INTIN(5-20) et les données du curseur dans INTIN(21-36). Ces données sont alors transmises par le VDI dans la table interne des données des curseurs. Un appel de la fonction de chargement ou de sauvegarde du menu redonne au curseur sa forme initiale, car la petite abeille est affichée pendant l'accès aux disquettes. Ensuite, le GEM choisit à nouveau la forme de flèche pour le choix des fichiers.

Si maintenant vous voulez insérer cette application dans un programme, il est naturellement inutile de transmettre le 'dessin' binaire du motif au tableau des données. Dans ce cas, il est conseillé, après la création de la forme finale du curseur, de transformer les deux 'dessins' binaires en nombres décimaux. Le programme a alors, à partir de la ligne 80, l'aspect suivant :

```
80  for i=0 to 31                    : rem début de la boucle
90  read x                           : rem lire la valeur
100 poke intin+10+i*2,x              : rem affecter
110 next i
120 vdisys 0                         : rem et exécution
130 end              : rem fin, on peut aussi mettre RETURN
140 rem — données du masque —
150 data 384, 2016, 8184, 65535, 32766, 62415, 62415
160 data 62415, 960, 960, 960, 960, 960, 960, 0, 0
170 rem — données du curseur —
180 data 0, 384, 2016, 8184, 29070, 24966, 384, 384
190 data 384, 384, 384, 384, 384, 0, 0, 0
```

Les données de cet exemple ne donnent pas le bonhomme de l'exemple précédent, mais une flèche verticale. Le point d'action a cependant ici pour coordonnées 8,0. Si vous préférez le bonhomme, vous pouvez obtenir les valeurs décimales correspondantes à l'aide du programme de conversion Binaire-Décimal du paragraphe 2.1.

Encore une remarque concernant ce programme : Il n'y a presque pas de limites à votre imagination, en ce qui concerne les différentes formes possibles du curseur. Verifiez cependant que vous respectez bien le format 16x16, sinon d'autres formes pourraient apparaître et non pas celle que vous aviez créées.

Il existe cependant encore un autre moyen pour changer la forme de la souris. Le GEM a en effet quelques formes de curseur prédéfinies. L'abeille, qui apparaît quand le calculateur travaille (par ex. au chargement), en fait partie. Cette forme ou une autre peut être choisie grâce à un appel de l'AES. Le prochain programme effectue ce choix. Nous reportons l'explication exacte de ce programme au chapitre sur la programmation des fenêtres et des menus, dans lequel sera présenté l'AES lui-même.

```
10  rem ** choix de la forme de la souris **
11  defdbl b,d : b=gb
12  cn=peek(b)
14  ii=peek(b+8)
20  d=peek(b+16)                     : rem déterminer le pointeur
50  poke cn,78                       : rem instruction
60  poke cn+2,1
70  poke cn+4,1
```

```
80  poke cn+6,1
90  poke cn+8,0
100 poke ii,4                 : rem choisir la forme
110 poke d,257                : rem mettre en place la forme
120 gemsys 78                 : rem et exécution
```

La valeur utilisée dans la ligne 100 indique la forme du curseur.

On a le choix entre :

0 Flèche
1 Curseur
2 Abeille
3 Main avec doigt pointé
4 Main plate
5 Réticule fin
6 Réticule épais
7 Réticule avec contour

En choisissant la fonction de chargement du BASIC ou une fonction analogue, la forme de la flèche ou de l'abeille est ici aussi rétablie.

4.2.4 Changement du type de caractère

Voici maintenant un autre exemple d'application, qui modifie cette fois-ci la police de caractères. Le GEM de l'ATARI peut en effet représenter de différentes façon les caractères. Outre les caractères standards, on peut utiliser des caractères soulignés, les caractères gras, les lettres 'creuses' et la représentation 'grise' que vous connaissez déjà avec l'éditeur BASIC. Pour pouvoir effectuer ces modifications, il nous faut à nouveau faire appel au GEM-VDI.

Voici un exemple de programme modifiant la police de caractères :

```
10  rem ** modifier l'écriture **
20  poke contrl,106           : rem code instruction
30  poke contrl+2,0           : rem nombre de paramètres
40  poke contrl+6,1
50  poke intin, 1+2           : rem écriture
60  vdisys 0                  : rem et exécution
```

```
70  print 'Quel text !'          : rem texte-éssai
80  poke contrl,106             : rem reprenons au début
90  poke contrl+2,0
100 poke contrl+6,1
110 poke intin,0                : rem écriture normale
120 vdisys 0                    : rem nouvelle exécutiom
```

Dans ce programme, la chaîne de caractères est écrite en caractères gras et 'illuminés' (gris). Après l'affichage du texte d'essai, on repasse en écriture standard pour éviter des difficultés éventuelles avec les commandes BASIC futures. Les lignes 10 à 40 sont semblables à celles de l'exemple précédent. La différence se trouve ici à la ligne 50, où est défini le type de caractère utilisé. Une addition n'est naturellement pas indispensable à cet endroit. Cette méthode facilite cependant le choix du type de caractère. Chaque mode d'écriture correspond en effet à un bit, ce qui permet une combinaison quelconque d'effets graphiques. La signification de chaque bit est la suivante :

Bit	Valeur	Caractères
0	1	gras
1	2	gris
2	4	obliques
3	8	soulignés
4	16	creux

Si, par exemple, l'écriture doit être oblique et grise avec des caractères soulignés, il faut affecter dans la ligne 50 la valeur 2+4+8, c'est à dire 14. D'après le nombre de combinaisons, vous pouvez de cette façon choisir 32 écritures différentes. Certaines d'entre elles comme par exemple l'écriture creuse inclinée (16+4=20), sont à vrai dire absolument illisibles, à moins de posséder une bonne dose de fantaisie.

4.2.5 Texte graphique

Avec les exemples ci-dessus, nous n'avons cependant pas examiné toutes les possibilités du VDI. Ainsi nous pouvons, à l'aide du GEM, formatter directement un texte et l'amener à une place quelconque de l'écran. De cette façon, il est possible d'afficher un texte en-dehors de la fenêtre Output du BASIC et même dans la ligne de menu. La fonction correspondante du VDI porte le numéro 11.

Voyons un exemple de programme qui utilise cette fonction :

```
10  rem ** texte graphique - Demo VDI **
15  texte$ = 'texte d'essai'
20  poke contrl,11              : rem code instruction
30  poke contrl+2,3             : rem nombre de paramètres
32  poke contrl+6,len(texte$)+2
34  poke contrl+10,10           : rem ID de la fonction
40  poke intin+2,1              : rem allongement du texte (0=off)
50  poke ptsin,50               : rem coordonnée X
60  poke ptsin+2,50             : rem coordonnée Y
70  poke ptsin+4,150            : rem longueur du texte suivant X
80  for i=1 to len(texte$)      : rem caractère ASCII
90  poke intin+2+i*2,asc(mid$(texte$,i,1))
100 next i
110 vdisys 0                    : rem et exécution
```

Ce programme affiche le texte contenu dans la variable TEXTE$ aux coordonnées 50,50 de l'écran, et allonge le texte jusqu'à qu'il atteigne la longueur indiquée de 150 caractères. Le 3, qui, dans la ligne 30, est affecté au vecteur Control, indique le nombre de paramètres à transmettre dans PTSIN. Les deux premiers sont les coordonnées du premier caractère dans le texte, 0,0 indiquant le point supérieur gauche. La longueur du texte suivant X est la largeur totale, que doit occuper le texte. Le texte est 'étiré' en insérant plusieurs blancs entre les caractères. Si l'on veut afficher le texte sans l'allonger, on peut aussi déconnecter le mode allongement, en affectant 0 dans la ligne 40.

Dans le programme est également intégré un peu de traitement de texte. Le VDI doit en effet savoir combien de caractères en tout il doit afficher. Ce nombre lui est communiqué ligne 32, à savoir par l'indication du nombre de paramètres qu'il doit lire dans le vecteur INTIN. La fonction LEN(TEXTE$) donne la longueur de la chaîne de caractères TEXTE$, le +2 est là pour INTIN(0) et INTIN(1). Puis le texte est affecté caractère par caractère dans le vecteur INTIN. Pour charger un caractère en mémoire, on doit écrire son code ASCII correspondant. ASCII signifie 'American Standart Code for Information Interchange' et est un code normalisé, qui est utilisé dans beaucoup d'ordinateurs et d'imprimantes. La fonction ASC('A') renvoie dans un programme BASIC la valeur ASCII du A, à savoir 65. Chaque caractère possède sa propre valeur, de telle sorte que dans la boucle de notre programme un texte quelconque peut être affecté à la variable TEXTE$. Ce caractère est extrait de la chaine de caractère avec la fonction MID$.

Mais nous allons reporter à plus tard l'étude du traitement de texte (paragraphe 7.4). Restons donc avec le VDI, dont le domaine d'application est le graphisme. Les instructions VDI les plus intéressantes sont expliquées dans le chapitre 6.1 à l'aide d'exemples. Pour avoir une vision globale des commandes VDI et AES, dont l'utilisation est cependant réservée en grande partie aux programmeurs C et langage machine, je conseille **"LE LIVRE DU GEM DE L'ATARI ST"**, qui est également distribué par **MICRO APPLICATION.**

Un bloc mémoire intéressant, dans lequel se trouvent certains paramètres du BASIC en liaison avec le GEM, est accessible directement grâce à la constante système SYSTAB qui est le pointeur de bloc mémoire. A l'aide de ce pointeur et des instructions PEEK et POKE, on peut obtenir des effets intéressants. En réalité, certains des paramètres accessibles ne se prêtent qu'à la lecture, car leur modification conduit facilement au plantage.

Le tableau suivant donne l'adresse et la signification de ces paramètres :

SYSTAB	résolution graphique(1=haute, 2=moyenne, 4=basse)
SYSTAB+2	type de caractère de travail de l'éditeur (voir plus loin)
SYSTAB+4	numéro AES d'accès à la fenêtre EDIT

SYSTAB+6	numéro AES d'accès à la fenêtre LIST	
SYSTAB+8	numéro AES d'accès à la fenêtre OUTPUT	
SYSTAB+10	numéro AES d'accès à la fenêtre COMMAND	
SYSTAB+12	flag EDIT	(0 : fermé, 1 : ouvert)
SYSTAB+14	flag LIST	(0 : fermé, 1 : ouvert)
SYSTAB+16	flag OUTPUT	(0 : fermé, 1 : ouvert)
SYSTAB+18	flag COMMAND	(0 : fermé, 1 : ouvert)
SYSTAB+20	pointeur du buffer graphique	
SYSTAB+24	flag GEM (0 = normal, 1 = off)	

La résolution graphique vaut 1 avec un écran monochrome et 2 (ou 4) avec un écran couleur. Un PEEK(SYSTAB) renvoie la valeur correspondant à cette résolution, cela peut de temps en temps être utile dans un programme graphique, car les coordonnées maximales de X et Y peuvent changer.

On entend par type de caractère de travail l'écriture 'grise' classique utilisée dans l'éditeur pour la ligne courante. Par exemple POKE SYSTAB+2,14 transforme cette écriture en une écriture romane. Une telle modification est à vrai dire nécessaire dans l'obscurité...

Les numéros AES d'accès aux différentes fenêtres de la 'tables de travail' BASIC sont les numéros que le GEM-AES a attribué à chaque fenêtre. Avec le numéro correspondant, on peut ensuite appeler l'AES, pour modifier (par ex. décaler, aggrandir ou diminuer) la position et la taille de la fenêtre choisie.

Les flags des différentes fenêtres contiennent l'état de celles-ci. Le GEM apprend en les testant si la fenêtre existe ou non. La manipulation de ces flags n'est pas sans danger, car, par exemple, une instruction LIST peut entrainer le plantage de l'appareil, si le flag LIST est 'éteint'. Ces flags ne devraient donc pas être modifiés.

Les quatres octets allant de l'adresse SYSTAB+20 jusqu'à l'adresse SYSTAB+23 contienne dans un mot double de 32 bits l'adresse du buffer graphique. Ce buffer garantit le contenu de la fenêtre OUTPUT si l'on modifie sa taille. En choisissant le point de menu Buffered Graphics, on peut choisir de ne pas utiliser cette mémorisation supplémentaire.

Ce faisant, on gagne 32 Koctets pour la mémoire de travail BASIC. Le buffer graphique prend en effet autant de place que la totalité de la mémoire écran. L'adresse de cette zone mémoire est obtenue en lisant le mot double mémorisé à partir de SYSTAB+20.

Il reste encore le flag GEM. Si on lui affecte 1, par un POKE SYSTAB+24,1, on déconnecte le GEM. On ne le remarque pas tout de suite, aucune fenêtre ne disparaissant, mais la modification d'une fenêtre n'est plus possible. De même, plus aucune entrée venant du clavier n'est acceptée. Le flag GEM ne doit donc être utilisé qu'à l'intérieur d'un programme, qui le remet à 0 pour les entrée/sorties et en fin de programme. Le seul avantage de mettre le flag à 1 réside dans le gain du temps que revendique normalement le GEM au processeur, pour réaliser la surveillance des fenêtres et des menus. En deconnectant le GEM, on interrompt cette surveillance et le processeur peut consacrer plus de temps à l'exécution du programme BASIC. Le programme se déroule donc plus rapidement, ce qui peut être très appréciable pour de longs calculs. L'accès aux disquettes est par ailleurs possible, car il est supervisé par le BIOS. Mais n'oubliez pas de reconnecter le GEM !

4.3 INTERPRETEUR/COMPILATEUR

Saviez-vous que l'ordinateur ne parle qu'un seul langage, à savoir le langage machine ? Vous allez maintenant me dire :' Mais l'ATARI ST connait aussi le LOGO et le BASIC'. Cela est également exact, mais à condition que ces langages de programmation aient été chargés dans l'ordinateur. Ce programme, par exemple BAS.PRG, est un traducteur, qui traite les instructions du BASIC en les transformant en code machine. Ce sont uniquement ces traducteurs qui font du ST un ordinateur capable de comprendre le BASIC, le LOGO ou le C.

On distingue deux types parmi de 'traducteurs' : les interprèteurs et les compilateurs. Pour le BASIC, il s'agit d'un interprèteur. Les programmes BASIC sont traduit ligne par ligne. En effet, seule la ligne de programme en cours d'exécution est interpretée. De ce fait, un des inconvénients des interprèteurs apparait déjà clairement : ils sont lents. Cela devient évident en examinant un exemple.

Prenons le cas d'une simple boucle :

```
10 FOR I = 1 TO 100
20 A = A * 3
30 NEXT I
```

Cette boucle triple 100 fois la variable A. A cet effet, l'interprèteur BASIC doit non seulement augmenter 100 fois la variable courante I et tester, si la boucle est terminée ou non, mais il doit encore ,chaque fois qu'il lit la ligne 20, interprèter cette dernière. Cette ligne est de plus placée à un endroit de la mémoire dont l'adresse doit être chaque fois évaluée. Par ailleurs, le 3 est une valeur, qu'il doit chaque fois transformer pour obtenir un format flottant. Tout ceci a certes lieu rapidement, mais de manière fréquente. C'est ainsi que le temps effectif de triplement de la variable A n'est qu'une petite partie du temps réellement utilisé.

A l'opposé des interprèteurs, il y a les compilateurs. Le langage de programmation C en fait partie. Le principe de la compilation permet à un programme C entièrement compilé d'être très rapide. Le programme, qui au début est représenté par un texte source, est traduit en une fois. Après ce processus, la compilation, on obtient un nouveau programme, qui a certes les mêmes fonctions que le programme source, mais qui est entièrement composé de code machine. De ce fait, un programme compilé est plus rapide. Cependant il est nécessaire de relancer une compilation à chaque modification. Un programmeur, qui est habitué à programmer en BASIC, va devoir, en passant aux langages compilés comme C, adopter une discipline de travail beaucoup plus grande. Le principe du Trial et Error de la mise au point d'un programme BASIC, c'est-à-dire commencer à entrer le programme, et de temps à autre lancer un RUN, demanderait ici beaucoup de temps. Le test d'un programme C demande non seulement du temps et donc de la patience, mais aussi des nerfs solides - car un programme machine peut, s'il y a une seule erreur, provoquer le plantage de l'appareil.

En réalité, un compilateur a aussi des inconvénients. La plupart du temps, toute une bibliothèque de routines d'aide est ajoutée au programme. Cette opération est effectuée lors de la compilation par le Linker.

Les routines d'aide, qui comprennent en C les fonctions d'entrée/sorties par exemple, offrent la plupart du temps beaucoup plus de possibilités que celles dont on a besoin. C'est pourquoi les programmes entièrement compilés sont plus longs que nécessaire. Et ces programmes sont effectivement longs ! Il suffit de regarder la taille de l'interprèteur BASIC, qui lui-même a été écrit grâce à un langage analogue au C. Si l'on écrivait ce programme directement en langage machine, il ne représenterait plus qu'une fraction de celui-ci. Mais la mise au point d'un programme est beaucoup plus compliquée en langage machine et coûte de ce fait beaucoup de temps.

On en arrive ainsi à la question : quel est la langage le mieux adapté dans tel ou tel cas ? Il n'y a pas de réponse valable dans tous les cas, car il faut tenir compte de beaucoup de facteurs. Le premier serait la vitesse à laquelle doit tourner un programme. Pour des tâches qui demandent un temps très court, par exemple pour la mesure en temps réel d'une grandeur variant constamment, un programme interprèté comme le BASIC n'est certainement pas particulièrement bien adapté.

D'un autre coté, le nombre probable de modifications à apporter au programme joue aussi un rôle. Si de plus, on veut employer l'ordinateur pour effectuer des calculs dont les résultats ne seront exploités qu'une seule fois, on a tout intérêt à écrire le programme dans un langage simple. Ecrire un programme C, le compiler et le linker représente pour une telle application une trop grande dépense.

En résumé, on peut dire qu'il est certainement idéal de maîtriser des langages des deux types. L'ATARI ST propose ici l'interprèteur BASIC et le compilateur C. Vous pourrez ensuite choisir, d'après les considérations précédentes, lequel de ces deux langages vous allez adopter pour résoudre tel ou tel problème particulier.

5. PREPARATION DU BUREAU

Avant d'entreprendre quoi que ce soit avec l'ATARI ST, on devrait d'abord définir l''environnement' dans lequel on souhaite travailler à l'avenir. Cela comprend le rangement correct du bureau, mais aussi la définition des différents paramètres des interfaces. Si l'on a règler tout cela, on peut, en choisissant le point de menu 'Save Desktop', sauvegarder l'état actuel de l''environnement' de travail sur disquette. Apres chaque Reset, cet état sera à nouveau rétabli, car le système d'exploitation charge le fichier DESKTOP.INF. Dans ce fichier se trouvent toutes les mises au point, qui ont été effectuées et sauvegardées. Les mises au point prévues ne permettent cependant pas toutes les configurations possibles. Mais on peut, en ce qui concerne la configuration des interfaces par exemple, aller plus loin que ce qui est proposé. Voyons cela de plus près :

5.1 CONFIGURATION DES INTERFACES

Si l'on veut définir le débit de transmission de l'interface série, on peut choisir une valeur comprise entre 300 et 9600 Baud. Ces valeurs correspondent aux débits dont on a besoin pour les communications séries. Mais il arrive de temps en temps que l'on souhaite effectuer une émission plus rapide (ou plus lente). Le chip implanté dans le ST, qui gère l'interface série, en est fort capable. Il maîtrise des vitesses allant de 50 jusqu'à 62500 Baud, ce qui est maintenant largement suffisant pour toutes les applications du port RS232.

On est cependant limité dans ce domaine par un autre facteur. En effet, le système d'exploitation ne supporte que des vitesses comprises entre 50 et 19200 Baud. Si vous voulez utiliser une vitesse, comprise entre ces dernières valeurs mais qui est en dehors du champ des vitesses habituel, vous devez le communiquer au système d'exploitation lui-même. Ceci n'est cependant possible qu'en langage machine. Nous devons donc écrire un petit programme machine.

Le plus simple serait ici d'utiliser un assembleur, que contient le kit de développement d'ATARI. Mais tout le monde ne possède pas ce programme, nous le ferons donc par le biais du BASIC.

Le programme BASIC suivant crée un programme machine nommé SETBAUD.PRG et le mémorise sur disquette. Pour l'appeler, il suffit de le faire à partir du Desktop de façon habituelle. L'interface série est ensuite configuré comme vous le désirez. Notons que l'appel du point de menu 'Set RS 232 Config.' remet immédiatement l'interface dans l'état initial.

Voici donc le programme :

```
10  rem ** configuration du nombre de bauds **
20  for i=1 to 72 step 2                : rem 36 mots
30  read a                              : rem lire
40  poke i+1999,a                       : rem placer
50  next i                              : rem en mémoire
60  bsave 'SETBAUD.PRG',2000,72         : rem et sauvegarder
70  end
80  rem — données du programme —
100 data 24602,0,44,0,0,0,0,0,0,0,0,0,0,0
110 data 16188,-1,16188,-1,16188,-1
120 data 16188,-1,16188,-1,16188,15
130 data 16188,15,20046,-8196,0,14
140 data 16999,16188,76,20033
```

Le programme ainsi créé définit une vitesse de transmission de 50 Bauds. Si l'on souhaite utiliser une autre vitesse, il faut remplacer le 15 en fin de la ligne 120 par la valeur correspondante.

Vous trouverez ces valeurs dans la table suivante :

Code	Débit correspondant (baud)
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
8	600
9	300

10	200
11	150
12	134
13	110
14	75
15	50

Tous les autres paramètres de l'interface, comme le protocole de transmission ou la parité, restent inchangés.

Si vous voulez en outre modifier le protocole de transmission, vous pouvez le faire en changeant le -1 de la ligne 120.

Voici la signification des différentes valeurs :

Code	Signification
-1	La valeur prédéfinie reste inchangée
0	Pas de Handshake
1	XON/XOFF
2	RTS/CTS
3	Les deux. N'a cependant pas de sens.

Encore une remarque concernant la reconfiguration de l'interface série : le système d'exploitation initialise, dans certaines fonctions, l'interface. Les paramètres initiaux qui ont été définis dans le menu sont alors à nouveau pris en compte. C'est en particulier le cas pour la vitesse de transmission. Si on remplace le nom du programme correspondant DESK1.ACC par DESK1.XXX par exemple, et qu'on réinitialise le système, on supprime cette possibilité. La vitesse de transmission qui a été définie dans notre programme est alors conservée.

Mais il y a encore une autre possibilité pour reconfigurer l'interface. Si vous voulez faire varier la vitesse de transmission de l'interface série dans un programme BASIC, vous pouvez le faire à l'aide de l'instruction CALL et d'un programme machine. CALL signifie appeler, c'est ce que fait effectivement cette instruction. Elle appelle un programme machine, qui doit cependant être prévu pour être utiliser à partir d'un programme BASIC. Dans le chapître 6.4, vous apprendrez grâce à des exemples comment combiner deux langages de programmation différents.

6. LES LANGAGES DE PROGRAMMATION

Pour un ordinateur du calibre de l'ATARI ST, un langage de programmation doit avoir un champ d'application très vaste, pour pouvoir exploiter toutes les particularités de l'ordinateur. Pour ces raisons, ce qui se fait de mieux, en ce qui concerne les langages, est tout juste assez bien pour l'ATARI. Les langages actuellement disponibles sont tous très confortables et recouvrent les principaux besoins des utilisateurs du ST. Quelques-uns de ces langages vont maintenant être examinés à la loupe, et nous allons voir quels sont les avantages et inconvénients de tel ou tel langage pour nos applications. Commençons par les interprèteurs LOGO et BASIC, qui sont très bien adaptés pour les tâches simples et par ailleurs très faciles à manier.

6.1 DR. LOGO

Le nom de ce langage de programmation, souvent écrit comme dans le titre, ne signifie pas qu'il s'agit d'un langage pour médecins. DR LOGO, comme il s'écrit en réalité, est l'abréviation de Digital Research LOGO et est aussi désigné par ATARI-LOGO dans le manuel de présentation. Ce programme, qui tourne aussi sous une forme semblable sur d'autres ordinateurs, est désigné à tord comme un langage pour enfants. En réalité, avec DR LOGO, on dispose d'un langage très confortable et très puissant.

LOGO a une petite similarité avec FORTH, que l'on considère volontiers comme un langage de programmation sans limites. La raison à cela est la possibilité de définir soi-même des instructions particulières. Ceci est également vrai pour LOGO.

Prenons un exemple.

Si l'on veut transmettre les données d'un programme à l'imprimante, cela est possible de manière simple. LOGO dispose uniquement des instructions COPYON et COPYOFF, qui dirigent toutes les tâches vers l'imprimante. Si l'on veut créer la commande LPRINT, bien connue en BASIC, on définit celle-ci soi-même très simplement.

Voici comment :

```
TO LPRINT :LIGNE
  COPYON PR :LIGNE COPYOFF
END
```

Un tel programme, qui est appelé par son nom, est une procédure. La procédure LPRINT[a] est maintenant tout aussi accessible que les autres instructions résidantes du LOGO. On imprime un texte simplement par LPRINT[Texte].

Cet exemple montre le principe de la programmation LOGO. Pour les tâches simples d'un programme, on écrit des procédures. Celles-ci sont alors de leur coté appelées par des procédures de rang supérieur et ainsi de suite, jusqu'à que tout le programme soit finalement réduit à une seule instruction. Cette structure hiérarchique a l'avantage de clarifier la programmation, mais exige aussi une certaine discipline en ce qui concerne l'élaboration du programme. Un programme construit de façon non systématique est plus tard très difficile à lire.

Un autre aspect de la philosophie du LOGO est l'orientation du langage vers les listes. Ainsi la plupart des informations sont gérées sous forme de liste, dont le traitement est assisté en grande partie par le langage. Les listes peuvent être composées d'un nombre quelconque de nombres mais aussi de textes. Grâce à des instructions spéciales, on peut effectuer des tris, des recherches ou des modifications. Ainsi, ce langage est très adapté au traitement de texte ou à la gestion de tables. Mais LOGO a encore une autre spécificité : le graphisme. On l'appelle aussi parfois Turtle-Graphik, car le curseur est représenté dans certaines versions du LOGO par une tortue. Dans l'ATARI ST, c'est un triangle, qui est mis en mouvement avec les commandes LEFT, RIGHT, FORWARD ou BACK. Si on le souhaite, il laisse un trait derrière lui, ce qui permet de réaliser facilement des graphiques.

Un exemple :

```
TO GRAPHIQUE
 FD 100 RT 111 FD 40
  TURN
  GRAPHIQUE
END

TO TURN
 FD 66 RT 66 FD 66
END
```

Cet exemple dessine avec l'instruction GRAPHIQUE un 'ornement' sur l'écran graphique. Comme la procédure GRAPHIQUE s'appelle elle-même, le programme tourne jusqu'à qu'on l'arrête ou qu'une coupure de courant survienne.

Cependant, une description complète du langage de programmation LOGO sortirait du cadre de ce livre. Pour ceux que cela intéresse, il y a un de très bons livres qui vous expliqueront plus en détail ce langage. Nous allons plutôt voir comment LOGO permet de contourner le système d'exploitation.

LOGO possède aussi ses propres instructions PEEK et POKE, elles portent les noms de .EXAMINE et .DEPOSIT . Par exemple, .EXAMINE 1102 donne l'adresse de la mémoire écran, alors que DEPOSIT 1262,0 commande l'impression de celle-ci.

6.2 BASIC ATARI

Le BASIC est sans doute le langage préféré des programmeurs. Cette préférence vient du fait que ce langage est facile à apprendre et est très souple en ce qui concerne le style de programmation. Un programme BASIC est vite écrit ; on trouve rapidement les erreurs grâce au principe Trial-and-Error qui consiste à corriger directement une erreur apparaissant lors de l'exécution. Cela n'est pas possible aussi facilement avec des langages comme le C ou le langage machine.

L'interprèteur du BASIC de l'ATARI mémorise les lignes introduites comme du texte. Ce texte est, lors de l'exécution, interpreté ligne par ligne. Certains interprèteurs travaillent un peu différemment. Ces derniers mémorisent chaque instruction BASIC sous forme de code, appelé Tokens. Cette méthode simplifie l'interprètation des instructions et augmente ainsi la vitesse d'exécution. Cependant le BASIC ATARI est, même sans la traduction en Tokens, si rapide que plus d'un programmeur BASIC expérimenté s'en étonne.

Un programme BASIC peut être chargé de la disquette directement dans un éditeur de texte pour y être modifié. Ceci n'est cependant pas indispensable, car l'éditeur BASIC offre un confort suffisant. Il est uniquement gênant que la fonction TAB ne soit pas autorisée. Par ailleurs, le BASIC ATARI se distingue encore par sa fonction Syntax-Test, très utile, qui vérifie immédiatement l'exactitude d'une ligne de programme. L'inconvénient réside dans le fait qu'on ne peut pas édité des textes normaux dans l'éditeur BASIC, sans que l'on obtienne, après l'introduction de chaque ligne, le message d'erreur 'Something is wrong'. C'est pourquoi il vaut mieux utiliser l'éditeur LOGO pour le traitement de texte.

A la différence des autres langages, dont on parle ici, le BASIC est orienté vers les lignes. Cela veut dire que chaque ligne a un numéro auquel peut accèder l'instruction GOTO ou GOSUB. Mais si l'on veut insérer plusieurs instruction dans le programme, il faut qu'il y ait une place suffisante. Pour cette raison, il est avantageux de choisir un intervalle de 10 lignes, en écrivant le programme. L'instruction AUTO, qui affiche automatiquement les numéros de ligne lors de la programmation, adopte également cet écart ,si il n'est pas déclaré différemment.

Si, malgré cela, on a plus de neuf lignes à insérer, une astuce peut servir. On renumérote les lignes avec l'instruction RENUM, en introduisant un écart suffisamment élevé entre les lignes. On peut ensuite insérer son morceau de programme, et enfin remettre une numérotation normale.

Le BASIC n'est pas un langage qui est prévu pour résoudre des problèmes particuliers. Alors que le COBOL et le FORTRAN sont respectivement prévus pour des tâches économiques (gestion) et scientifiques, le BASIC peut être utiliser pour tout. Il n'offre certes pas d'instructions spéciales, comme le tri ou le calcul statistique, mais on peut les créer dans des sous-programmes. Plus problématique est cependant la gestion de processus très rapides. Pour de telles tâches, critiques du point de vue temps, le langage machine est beaucoup plus adapté. On peut bien sûr combiner ces deux langages, c'est ce que nous verrons au chapître 6.4.1.

Ce livre est également écrit pour les programmeurs BASIC qui veulent utiliser pleinement les possibilités 'fantastiques' de l'ATARI ST. Ce n'est, il est vrai, pas possible dans tous les cas, mais le BASIC offre, avec les instructions systèmes PEEK, POKE, CALL ou USR, la possibilité de faire appel au système d'exploitation ou à des programmes en langage machine. Pour ces raisons, le livre ne s'intitule pas 'EXAMINEs et DEPOSITs' (c'est comme PEEK et POKE sont appelés en LOGO), car seul le BASIC offre à la fois une programmation simple et la possibilité d'exploiter tout le système.

6.3 'C'

Le langage de programmation C fait partie des langages évolués. Son curieux nom provient de ce que les créateurs de ce langage ont appelé leurs versions successives avec des lettres de l'alphabet. Au début, il y eu A, puis B et enfin C, un langage actuellement très répendu, qui est utilisé dans un grand nombre d'ordinateurs.

C est un langage très proche de la machine. Ceci se manifeste en premier lieu par le fait que les routines d'entrée/sortie n'appartiennent pas au noyau central de ce langage. La raison à cela est que ces routines sont spécifiques au système et dépendent donc de l'ordinateur lui-même. Malgré cela, on peut naturellement effectuer des opérations d'entrée/sortie. A cet effet, l'ensemble fourni avec le compilateur C possède une bibliothèque, contenant toutes ces fonctions spécifiques. Le noyau central auquel on a ajouté les routines de la bibliothèques permet de réaliser toutes les applications possibles. Le lien est effectué par le Linker. Ce programme utilitaire est également disponible avec tout compilateur C.

L'avantage de cet ensemble d'instructions plutôt spartiate qu'offre le langage C, est que les programmes C sont transportables d'un ordinateur à un autre, en ayant un nombre minime de modifications à effectuer. Ainsi, un programme, ou même un système d'exploitation, écrit en C sur un ordinateur particulier, est compatible avec n'importe quel autre d'ordinateur.

L'ATARI ST a tiré profit de cet avantage, puisque le GEM, qui est contenu dans le système d'exploitation, a aussi été écrit en C. La même version du GEM existe également sur quelques autres ordinateurs, comme par exemple l'IBM PC.

Le puissant système d'exploitation UNIX, qui trouve surtout son application dans les gros systèmes, est également écrit en C. On voit donc que ce langage est très intéressant !

Nous n'avons hélas pas suffisamment de place pour une description plus détaillée du C, car la philosophie C est très complexe. Je me propose donc simplement de vous donner un petit aperçu de ce langage de programmation.

C appartient à la famille du langage ALGOL. De cette famille sont également nés FORTRAN et PASCAL ; ces langages se distinguent par leur haut degré de lisibilité et par leur syntaxe structurée. Une des caractéristique du langage C est la notion de procédure, dont l'appel est considéré comme une instruction. La procédure PRINTF en fait partie. Elle appartient aux routines standards d'entrée/sortie, qui sont disponibles dans la bibliothèque citée plus haut. Cette procédure affiche, de la même façon que l'instruction PRINT en BASIC, un texte ou des valeurs numériques. Les chaînes de caractères ou les valeurs numériques, devant êtres affichées, sont transmises à cette fonction, ainsi que l'indication du format dans lequel elles doivent être affichées. En C, la transmission des paramètres se fait de la manière suivante :

```
PRINTF ('%5.2f %6.1f\n',A,B)
```

Les arguments, qui commencent par un %, précisent le format de l'affichage. %5.2f signifie ici, que la valeur de la variable A sera représentée avec au moins 5 chiffres avant la virgule et 2 chiffres après la virgule. Le f indique ici la représentation en virgule flottante (floating point).

Aussi compliquée que cette expression apparaisse, elle n'en est pas moins confortable. Pour cette raison, elle est utilisée fréquemment.

Mais revenons au noyau central du C. Un programme C ne contient que des structures de contrôles simples, comme les conditions, les boucles, les réunions et les sous-programmes. Les variables doivent être déclarées explicitement au début du programme, afin de déterminer leurs types. Une programmation simple et rapide comme en BASIC n'est ici pas permise, mais cela facilite la recherche des erreurs.

La structure d'un programme C est la suivante: on déclare d'abord les variables nécessaires au programme. Puis on écrit chaque procédure, qui sont toutes séparées les unes des autres. Pour pouvoir exécuter un programme C, on cherche d'abord la procédure qui porte le nom MAIN(). Il s'agit ici du programme proprement dit, qui contient habituellement une série d'appels de procédures. Comme l'appel de ces procédures finit la plupart du temps par un retour à la procédure MAIN(), on a aisément un aperçu de la structure globale de l'ensemble du programme.

Les petits programmes sont, la plupart du temps, uniquement constitués de la routine MAIN(). Voyons un petit programme de ce type :

```
/* conversion rayon-périmètre */

main()
 {      int debut, fin, step;
        float rayon, perimetre;

        debut = 1;          /* rayon initial */
        fin = 10;           /* rayon final */
        step = 1;           /* pas */

        rayon = debut;
        while (rayon <= fin)
        {
```

```
            perimetre = 2 * rayon * 3.141592;
            printf('%2.0f %7.3f\n',rayon,perimetre);
            rayon = rayon + step;

        }

    }
```

La première ligne est un commentaire. Un commentaire est délimité par les séquences de caractères '/*' et '*/', et est ignoré par le compilateur. Notre programme commence avec la procédure MAIN. Les parenthèses vides indiquent qu'aucun paramètre n'est transmis à cette procédure.

Puis on déclare les variables en indiquant leurs types.

On dispose des différents types suivant :

int	nombre entier
float	nombre réel (virgule flottante)
char	caractère (un seul)
short	entier court
long	entier long
double	réel double-précision

Ensuite, on initialise les variables. L'instruction de contrôle qui suit est une boucle calculant 10 fois le périmètre. C'est une boucle conditionnelle WHILE qui est répétée aussi longtemps que la condition est vérifiée. Il n'y a pas ici de WEND, comme en BASIC, pour indiquer la fin de la boucle WHILE. C'est la séquence d'instructions entre accolades qui est ici reprise par la boucle.

Cette séquence calcule le périmètre d'un cercle à partir du rayon et affiche les deux valeurs sous un certain format. Puis le rayon est augmenté de 1 et on recommence la boucle tant que la valeur finale 10 n'est pas dépassée. Autrement, l'exécution du programme est terminée.

Comme on le voit, un programme C n'a rien de très particulier, si l'on le compare avec un autre langage évolué. En C, toutes les instructions sont suivies d'un point-virgule, comme en PASCAL. Les instructions IF, ELSE, WHILE or FOR sont classiques et bien connues. Un programme C est, par sa structure, relativement facile lire, et le temps passé à l'écrire est récompensé la très grande vitesse d'exécution du programme compilé.

Avec l'ATARI ST, il y a encore une autre raison incitant à utiliser le C. Le GEM a en effet été programmé en C. Le dévellopement de programmes graphiques en C est donc très simple. Dans le kit de dévellopement de l'ATARI, se trouve une immense bibliothèque d'instructions GEM et de paramètres systèmes, qui rendent toutes les possibilitées du GEMDOS accessibles au programmeur C.

6.4 LE LANGAGE MACHINE DU MC 68000

Parmi tous les langages de programmation, le langage machine joue un rôle particulier. Ce langage constitue à la fois le plus primitif et le plus puissant moyen de programmer un ordinateur. Cette contradiction apparente est simple à expliquer :

Dans chaque ordinateur se trouve une unité centrale de calcul, la CPU (Central Processor Unit). Ce composant, le 'cerveau' de l'ordinateur, abat tout le travail dont l'ordinateur est chargé. Si l'on programme dans un langage évolué comme BASIC ou LOGO, les instructions que l'on a entrées sont d'abord traduites dans un langage que le microprocesseur est capable de comprendre, à savoir le langage machine. L'avantage, que le langage machine possède sur tous les autres langages, est le fait qu'il peut diriger toutes les fonctions de l'ordinateur. C'est justement ici que se situent les limites d'un langage évolué comme par exemple le BASIC. On est alors obligé, pour certaines opérations, de faire un détour par le domaine machine.

Nous sommes déjà tombés sur une de ces limites dans le chapitre, où il était question de reconfigurer soi-même l'interface série. Pour réaliser cela, même l'instruction à tout faire POKE n'était pas utilisable, il a donc fallu écrire un programme machine. Mais cela ne représente que la pointe de l'iceberg.

Un autre avantage de la programmation directe du processeur réside dans la vitesse qui peut ainsi être atteinte. Il suffit, pour s'en rendre compte, de regarder la rapidité avec laquelle l'écran est effacé quand on appelle la routine correspondante.

Un programme en BASIC équivalent aurait l'aspect suivant :

```
10 defdbl a,b : b = 1102
20 a = peek(b)
30 for i = a to a + 32768
40 poke i,0
50 next i
```

Quelqu'un qui est capable de mesurer visuellement la vitesse d'un programme BASIC, va tout de suite se rendre compte que même un BASIC rapide comme celui du ST nécessite quelques secondes pour exécuter ce programme. En effet, on peut observer que l'écran est effacé ligne par ligne. Par contre, le processus d'effacement que met en oeuvre le système d'exploitation va si vite qu'on ne peut suivre son déroulement.

Certains vont sans doute penser que de tels processus sont gérés par le système d'exploitation, et donc qu'il n'est pas nécessaire de se préoccuper de ces problèmes de temps. Mais qu'en est-il d'une application comme par exemple le traitement de texte, où un processus de tri ne doit pas obligatoirement durer le temps d'une pause café !

Vous voyez donc qu'on ne peut laisser de côté le langage machine, si on veut exploiter l'ensemble du système, surtout avec un microprocesseur aussi puissant que le MC 68000 qui offre, en relation avec le système d'exploitation GEMDOS, d'énormes possibilités.

Examinons donc tout d'abord le processeur et son langage. Ici, il n'existe pas de variables ou d'instructions aussi confortables que PRINT 16^3. Bien que le programmeur dispose de quelques instructions, celles-ci ne s'appliquent qu'aux bits, octets, mots, mots longs et registres. Si l'on veut mémoriser une valeur, il faut déterminer son adresse, l'y écrire, et se souvenir de l'adresse en question...

Cependant, pas de panique : ce n'est pas aussi compliqué que ça en a l'air. Examinons un exemple et supposons que les adresses $1000 et $1002 contiennent deux mots (je rappelle que 1 mot = 2 octets = 16 bits) qui doivent être additionnés. Le résultat est ensuite stocké à l'adresse $1004. Un programme machine réalisant cette opération a l'allure suivante :

```
MOVE.W $1000,D0
ADD.W  D0,$1002
MOVE.W D0,$1004
```

C'est fait ! L'instruction MOVE déplace, comme son nom l'indique, des données dans l'ordinateur. Le suffixe .W indique qu'il s'agit ici d'un mot, donc de 16 bits. Deux autres suffixes sont possibles: .B pour un octet (Byte) ou .L pour un mot long. La première instruction prend le mot contenu à l'adresse mémoire $1000 et le place dans D0. D0 est un registre donnée, c'est-à-dire une case mémoire de 32 bits propre au CPU. Le MC 68000 en possède 8: D0 jusqu'à D7. Dans ces registres, on stocke en général les données qui sont en cours de traitement. Les registres ont encore d'autres avantages par rapport à la mémoire de travail, que nous ne développeront cependant pas pour l'instant.

Le registre donnée D0 contient donc maintenant le contenu de l'adresse $1000. Puis, grâce à l'instruction ADD.W, on additionne à ce registre le contenu de l'adresse $1002. On retrouve ici le suffixe .W indiquant la taille de la donnée. Le résultat de l'addition se trouve alors en D0. Il ne reste plus qu'à charger ce résultat à l'adresse voulue. On emploie pour cela à nouveau l'instruction MOVE.W, en intervertissant simplement le registre et l'adresse mémoire. L'instruction MOVE transmet toujours la valeur de l'adresse mémoire (ou registre) indiquée avant la virgule vers celle indiquée après la virgule. Une syntaxe générale serait :

MOVE.Type-de-donnée Source,But

On entend par Type-de-donnée la lettre .B, .W ou .L ; La source et le but indiquent d'où provient la donnée et où elle doit être placée. Il existe différentes possibilités pour adresser la mémoire. Dans le tableau suivant, on trouvera les différents modes d'adressage du MC 68000 :

MODE D'ADRESSAGE	EXEMPLE
Direct à registre :	
- direct à registre donnée	CLR D0
- direct à registre adresse	MOVEA A1,A6
- direct au registre d'état	MOVE D0,SR
- direct au registre condition	MOVE D0,CCR
Immédiat :	
- immédiat long	MOVE#$20000,D0
- immediat court	MOVE #$20,D0
- immédiat rapide	MOVE #9,D0
Absolu :	
- absolu court	NOT $2000
- absolu long	NOT $18000
Indirect avec registre d'adresse :	
- simple	CLR(A0)
- avec distance	CLR 80(A0)
- avec distance et index	MOVE8(A0,D0),$40
- avec prédécrémentation	CLR -(A7)
- avec postincrémentation	CLR (A7)+
Relatif au compteur ordinal :	
- avec distance	MOVE 4(PC),A0
- avec distance et index	MOVE8(PC,A0),D0
- instruction Branch	BRA label

Comme le montre le tableau ci-dessus, les modes d'adressages se divisent en 5 catégories. Les 3 premières sont facile à expliquer.

Direct à registre :
on adresse ici directement un registre comme source ou comme but, dont on déplace ou modifie le contenu. Les registres possibles sont les registres de donnée D0 à D7, les registres d'adresse A0 à A7, le registre d'état SR et le registre de code condition CCR. Utilisez A7 avec prudence, car ce registre d'adresse contient le pointeur de pile (Stackpointer).

Immédiat :
ici, on affecte une constante à une adresse ou un registre. Ce mode d'adressage n'a de sens que pour la source, car comment le processeur peut-il modifier une constante ? Ce mode d'adressage correspond à peu près à l'instruction POKE, car MOVE #10,1000 tout comme POKE 1000,10 place une valeur fixe (10) à une adresse (1000).

Absolu :
on pointe directement l'adresse voulue pour utiliser son contenu. Les instructions PEEK et POKE ne peuvent manipuler que des adresses absolues. Le langage machine du MC 68000, quant à lui, offre encore d'autres modes d'adressage. Ces modes d'adressage sont, à vrai dire, un peu difficiles à comprendre, mais sont en contrepartie très efficaces.

Voyons à présent les deux autres types d'adressage, en commençant par l'adressage indirect :

Ici, on n'utilise pas le registre lui-même, mais son contenu interpreté comme une adresse. Ce mode d'adressage permet d'exploiter très facilement les pointeurs, en chargeant la valeur du pointeur dans un registre et en se servant de ce dernier pour adresser indirectement.

Un exemple :

```
MOVE.L 1102,A0    * mettre l'adresse de la mémoire écran dans A0
MOVE.W #1,(A0)    * et afficher un point à l'écran
```

Ce minuscule programme machine est également réalisable en BASIC, mais nécessite alors un plus grand nombre d'instructions. Le programme BASIC équivalent a l'allure suivante :

```
10 defdbl a,b       : rem définir des mots longs
20 b = 1102         : rem adresse du pointeur
30 a = peek(b)      : rem trouver l'adresse de la mémoire écran
40 poke a,1         : rem et afficher le point
```

L'avantage de l'adressage indirect est mis en évidence par cet exemple simple. On peut, en outre, ajouter à l'adresse une distance, appelée offset. Le processeur additionne alors cet offset à l'adresse contenue dans le registre, et considère le résultat comme adresse effective. Dans l'exemple ci-dessus, on peut utiliser cette méthode pour modifier (ou lire) non pas le premier mot de l'écran mais le n-ième. La syntaxe serait donc MOVE.W #1,n(A0), où n représente l'offset. Si le point doit, par exemple, se situer sur la deuxième ligne, on utilise MOVE.W 1,80(A0). La distance (offset) ne peut cependant pas dépasser la valeur +/- 32767. Cela permet tout de même de traiter tout l'écran.

L'offset est donc seulement une constante. Si l'on veut faire varier cette distance dans le programme, on peut encore ajouter un registre, dont le contenu sera également additionné à l'adresse. La constante offset a alors pour valeurs limites +/-127, le registre pouvant quant à lui contenir un mot long. Ce mot ou mot long est signé ; cela signifie que le bit de plus fort poids indique le signe de la donnée (1=négatif, 0=positif). Grâce à ce mode d'adressage, on peut adresser toute la mémoire de l'ATARI ST, ce qui n'est à vrai dire pas indispensable. Ce mode d'adressage devient intéressant quand on traite une grande quantité de données, dont la position dans la mémoire est variable.

Examinons à présent l'adressage indirect avec postincrémentation et prédécrémentation. Ces deux derniers termes, qui se prononcent difficilement, signifient que le registre choisi est augmenté (incrémenté) resp. diminué (décrémenté) après (post) resp. avant (pre) son utilisation. Ce mode d'adressage est très important et intervient dans chaque programme. De cette manière, on peut très facilement effectuer des opérations sur la pile. La pile représente en effet une suite de données et d'adresses, auxquelles on a souvent accès, comme par exemple lors de l'instruction RTS (comparable avec RETURN en BASIC), qui prend sur le dessus de la pile l'adresse de retour du programme appelant.

Les programmeurs de processeurs 8 bits, comme le 6502 qui est implanté dans l'ATARI XL ou C64, utilisent les instructions PLA et PHA pour accéder à la pile. Mais ces deux instructions permettent uniquement de prendre un octet du dessus de la pile resp. déposer un octet sur la pile.

Pour cette même opération, à savoir l'utilisation de la pile, le 68000 de l'ATARI ST propose l'adressage avec postincrémentation et avec prédécrémentation, qui en outre permettent l'utilisation de mots et de mots longs. Si l'on veut prendre un mot du dessus de la pile, on utilise l'instruction MOVE.W (SP)+,D0. Le mot se trouve alors dans le registre donnée D0, et le pointeur de pile SP (stack-pointer) est augmenté de la taille (en nombre d'octets) d'un mot. Si l'on exécute encore une fois la même instruction, on obtient évidemment la valeur suivante dans la pile, sans avoir pour cela été obligés de modifier explicitement le pointeur de pile.

La même chose est valable pour déposer des données sur la pile. Mais comme le stack-pointer (pointeur de pile) pointe toujours sur la donnée déposée en dernier lieu, il faut, avant de déposer la prochaine donnée, diminuer le pointeur du nombre d'octets correspondant à la taille de la donnée. Pour cela, on utilise l'adressage avec prédécrémentation. Le dépot d'un mot long sur la pile peut se faire avec MOVE.L #$F8000,-(SP) . Le stack-pointer est alors diminué de 4.

Le dernier mode d'adressage a pour intitulé: adressage relatif par rapport au compteur ordinal. Ce mode adressage dépend donc du compteur ordinal (compteur de programme), qui contient l'adresse momentanée de l'instruction en cours d'exécution. Il est ainsi possible de créer des tables et de les lire, bien que l'adresse réelle du programme soit éventuellement variable, et le cas échéant inconnue. Ce mode d'adressage rend également le microprocesseur 68000 capable de charger des grands programmes à des adresses quelconques en mémoire. Cela simplifie entre autre la combinaison du BASIC et du langage machine, car on peut par exemple 'affecter' un programme à une chaîne de caractères, et l'exécuter ensuite. Ce procédé très confortable et pratique n'est pas faisable, de manière simple, dans les ordinateurs ne possédant pas ce mode d'adressage, car le programme doit accéder à des variables particulières et ne sait pas où il se trouve lui-même.

La syntaxe des instructions avec adressage relatif au compteur ordinal est semblable à celle faisant intervenir l'adressage indirect avec registre. Il suffit de remplacer le registre par le PC (Program-Counter).

L'instruction Branch représente encore une nuance de l'adressage relatif. Ici, on ne pointe pas des données, mais le programme lui-même. Par une instruction Branch, on branche le programme à une adresse déterminée, sans condition s'il s'agit de l'instruction BRA (Branch Always) ou avec condition sinon. Les branchements conditionnels sont nombreux. La plupart du temps, une instruction Branch conditionnelle est précédée par une comparaison entre deux données.

Par exemple :

```
CMP.W #10,D0
BLT   PLUSPETIT
...
```

On branche ici le microprocesseur sur la partie du programme commençant à l'adresse 'PLUSPETIT' (label), à condition que le contenu du registre D0 soit plus petit que 10. BLT est l'abréviation de 'Branch if Less Than'. On peut donc comprendre la signification d'une instruction Branch rien qu'en lisant sa mnémonique.
Le tableau suivant contient toutes les instructions Branch possibles :

Mnémonique	**condition en anglais**	**condition en français**	**condition sur les flags**
BEQ	equal	égal	Z=1
BNE	not equal	inégal	Z=0
BPL	plus	positif	N=0
BMI	minus	négatif	N=1
BGT	greater than	supérieur	Z+(N&V)=0
BLT	less than	inférieur	N&V=1
BGE	greater or equal	supérieur ou égal	N&V=0
BLE	less or equal	inférieur ou égal	Z+(N&V)=1
BHI	higher	supérieur (non signé)	C+Z=0

BLS	lower or same	inférieur ou égal (non signé)	C+Z=1
BCS	Carry set	Carry à 1	C=1
BCC	Carry clear	Carry à 0	C=0
BVS	overflow	dépassement	V=1
BVC	no overflow	pas de dépassement	V=0
BRA	branch always	branche toujours	
BSR	branch to subroutine	branchement à un sous-programme	

Le signe +/- correspond ici à un OU logique, alors que & représente un ET logique (voir paragraphe 3). Les lettres dans la colonne de droite correspondent aux flags (un flag = un bit) du registre code-condition (CCR). Chacun de ces bits a une signification particulière et est éventuellement modifié par les opérations de comparaison.

Comme nous parlons justement des branchements et du déroulement d'un programme, nous pouvons dés à présent examiner toutes les instructions de saut. Certaines d'entre elles branchent à des sous-programmes, qui une fois exécutés renvoient le processeur au programme appelant, là où le branchement a eu lieu. On connait déjà l'instruction JSR ou BSR, car JSR (Jump to Sub-Routine) est très similaire à l'instruction BASIC GOSUB. L'instruction JMP (Jump) nous est aussi familière, car son pendant en BASIC est GOTO. Cependant, le microprocesseur connait d'autres instructions de branchements.

Il y a entre autre l'instruction TRAP, qui joue un grand rôle, en particulier dans l'ATARI ST. Avec cette instruction, on active le GEMDOS ou le BIOS, suivant le numéro qui suit le TRAP. TRAP #1 réveille le GEMDOS, TRAP #13 le BIOS et TRAP #14 le BIOS étendu, c'est-à-dire le XBIOS. Un tel appel de fonction est appelé Exception ou Interruption software. Le numéro de la fonction appelée est déposé sur la pile, ainsi que d'éventuels paramètres. Quand la routine correspondante du système d'exploitation a été entièrement exécutée, le programme appelant reprend le contrôle de l'ordinateur. Celui-ci doit alors reprendre de la pile tous les paramètres qu'il y a déposés, pour que la pile soit mise à jour. Cela consiste généralement à modifier le pointeur de pile SP en lui additionnant le nombre d'octets adéquat.

Voici la liste de toutes les instructions de branchements du MC 68000 :

MNEMONIQUE	SIGNIFICATION
Bcc	Branchement conditionnel
BRA	Branchement inconditionnel
BSR	Branchement à un sous-programme
CHK	Examen d'un registre donnée aux limites
DBcc	teste une condition, décrémente et branche
JMP	Saut à une adresse
JSR	Saut à un sous-programme
NOP	Pas de fonction (No opération)
RESET	Remise à zéro de tous les composants périphériques
RTE	Retour d'une exeption (interruption software)
RTR	Retour avec chargement des flags
RTS	Retour d'un sous-programme
Scc	Modification d'un octet suivant le résultat du test
STOP	Interrompre le traitement
TRAP	Saut à une exeption
TRAPV	Saut à une exeption si le flag V est à 1

Dans le petit exemple, nous avons effectué une opération arithmétique, à savoir l'addition. A l'opposé de celle-ci, il existe aussi la soustraction. Pour ces opérations, un processeur - 8 - bit classique, comme le 6502, "s'essouffle" énormément. Ce n'est pas le cas du MC 68000, qui en outre possède d'autres opérations. Il peut en effet effectuer des multiplications et des divisions, toutes ces opérations pouvant s'appliquer à des données de taille différente.

Voici donc une vue d'ensemble des opérations arithmétiques qui s'appliquent à des nombres entiers :

MNEMONIQUE	SIGNIFICATION
ADD	Addition binaire
ADDA	Addition binaire d'adresses
ADDI	Addition avec une constante

ADDQ	Addition rapide d'une constante
ADDX	Addition avec retenue
CLR	Effacement d'un opérande
CMP	Comparaison de deux opérandes
CMPA	Comparaison de deux adresses
CMPI	Comparaison avec une constante
CMPM	Comparaison de deux opérandes adressés suivant le mode (Ax)+
DIVS	Division signée
DIVU	Division non signée
EXT	On complète l'opérande avec le signe
MULS	Multiplication signée
MULU	Multiplication non signée
NEG	Opposé d'un opérande (complémentation à deux)
NEGX	Opposé d'un opérande avec retenue
SUB	Soustraction binaire
SUBA	Soustraction binaire d'adresses
SUBI	Soustraction d'une constante
SUBQ	Soustraction rapide d'une constante
SUBX	Soustraction avec retenue
TST	Test de la nullité et du signe de l'opérande

De plus, le processeur peut, pour le traitement des nombres entiers, travailler directement avec des nombres codés en BCD.

Il dispose pour cela des instructions suivantes :

MNEMONIQUE	SIGNIFICATION
ABCD	Addition de deux nombres BCD
NBCD	Négation d'un nombre BCD (complémentation à neuf)
SBCD	Soustraction de deux nombres BCD

Voyons à présent les opérateurs logiques, que l'on connait déjà en partie avec le BASIC :

MNEMONIQUE	SIGNIFICATION
AND	ET logique
ANDI	ET logique avec une constante
EOR	OU exclusif
EORI	Ou exclusif avec une constante
NOT	Inversion d'une opérande
OR	OU logique
ORI	OU logique avec une constante
TAS	Teste et met à 1 un bit spécial

On peut également manipuler les bits d'un opérande.

On dispose pour cela des instructions suivantes :

MNEMONIQUE	SIGNIFICATION
BCHG	Inversion d'un bit précis
BCLR	Mise à 1 d'un bit précis
BSET	Mise à 0 d'un bit précis
BTST	Teste l'état d'un bit précis

Le microprocesseur peut encore effectuer des opérations de décalage et de rotation sur les bits d'une opérande :

MNEMONIQUE	SIGNIFICATION
ASL	Décalage arithmétique vers la gauche (*2)
ASR	Décalage arithmétique vers la droite (/2)
LSL	Décalage logique vers la gauche
LSR	Décalage logique vers la droite
ROL	Rotation vers la gauche
ROR	Rotation vers la droite
ROXL	Rotation vers la gauche avec la retenue
ROXR	Rotation vers la droite avec la retenue

Nous arrivons maintenant aux instructions qui déplace des données à l'intérieur de l'ordinateur :

MNEMONIQUE	SIGNIFICATION
EXG	Echange du contenu de deux registres
LEA	Charge une adresse effective
LINK	Montage de la pile
MOVE	Transfert d'une donnée
MOVE from SR	Transfert du contenu du registre d'état
MOVE to CCR	Chargement des flags
MOVE to SR	Chargement du registre d'état
MOVE USP	Chargement du pointeur de la pile utilisateur
MOVEA	Transfert d'une adresse
MOVEM	Transfert de plusieurs registres
MOVEP	Transfert de données vers les composants périphériques
MOVEQ	Transfert rapide d'une constante
PEA	Dépose une adresse sur la pile
SWAP	Intervertit les deux moitiés d'un registre
UNLK	Démontage de la pile

Voici donc toutes les instructions du MC 68000. Par leur combinaison avec les différents modes d'adressage, on dispose d'un très grand nombre d'instructions, afin de rendre un programme aussi efficace que possible.

Nous allons maintenant examiner le programme que nous avons utiliser pour ramener la fréquence de l'image RGB à 50 Hz. La séquence d'instructions machines, appelées aussi mnémoniques, qui constitue ce programme, est celle-ci :

```
CLR.L    -(SP)
MOVE.W   #$20,-(SP)
TRAP     #1          * Set supervisor-state
ADDQ     #6,A7       * réparer la pile
MOVE.B   #2,$FF820A  * mettre en place la fréquence PAL-SECAM
MOVE.L   D0,-(A7)    * ancien SPP
MOVE.W   #$20,-(SP)
TRAP     #1          * set user-state
```

```
ADDQ.L   #6,SP      * réparer la pile
CLR.W    -(SP)
TRAP     #1         * retour au Desktop
```

Le premier TRAP ordonne au GEMDOS de mettre le système en mode superviseur. Cela est nécessaire, car l'accès au domaine des I/O est privilégié et entraînerait, en mode utilisateur, une Bus-Error (et deux champignons atomiques...). Après la correction du pointeur de pile, on remplace la valeur de la fréquence par 50 Hz. Puis, par l'instruction TRAP #1, on appelle à nouveau le GEMDOS avec le même numéro de fonction que précedemment ($20). Cette fois-ci, on transmet à la fonction l'ancien pointeur de la pile superviseur, qui avait été placé dans D0 lors du premier appel. Enfin, l'ordinateur est remis en mode utilisateur. Après une autre réparation de la pile, on appelle la fonction 0 du GEMDOS, qui rend la main au programme appelant.

Le programme BASIC générant le code machine contient, en outre, quelques valeurs qui sont nécessaires au programme utilitaire généré. Ces valeurs n'appartiennent pas au programme lui-même, mais sont indispensables afin que le GEMDOS puissent connaître la longueur et le type du programme. Si nous appelons, à partir d'un programme BASIC, un programme machine grâce à l'instruction CALL, ces informations supplémentaires ne sont pas nécessaires. Tout aussi inutile, et même faux, serait l'utilisation de l'instruction TRAP #1 pour appeler la fonction 0, car,après cela, le GEMDOS ne rendrait pas la main au BASIC, mais au Desktop. Un programme appelé par un CALL se termine généralement par un RTS, qui redonne effectivement le contrôle au BASIC.

6.4.1 Combinaison avec le BASIC

Les avantages du langage machine sont donc si importants, qu'il serait certainement intéressant de le combiner avec un langage bien plus facile à écrire tel que le BASIC. Parmi les instructions BASIC qui accède au domaine machine, nous connaissons déjà PEEK et POKE. Mais il nous manque encore la possibilité d'appeler directement des routines machines à partir d'un programme BASIC. L'instruction CALL est prévue à cet effet. Cet appel machine offre en outre la possibilité de transmettre des paramètres au programme machine, ce qui permet d'obtenir une programmation très souple.

Examinons tout d'abord l'appel CALL lui-même. Pour cela, nous allons considérer un exemple qui permet de configurer la vitesse en bauds de l'interface série à partir d'un programme BASIC. Dans le chapitre sur la configuration des interfaces, nous avons déjà étudié un programme analogue, qui cependant ne pouvait être appeler que du Desktop.

Regardons donc ensemble l'autre version :

```
10  rem ** configuration de l'interface série **
12  a$=space$(40)                  : rem réserver la mémoire
14  b=peek(varptr(a$)+2)           : rem déterminer l'adresse
20  for i=0 to 36 step 2           : rem dans une boucle
30  read a                         : rem lire les données
40  poke b+i,a                     : rem et les charger
50  next i
60  input 'code-débit ';x          : rem saisir le code
70  poke b+22,x                    : rem insérer
80  call b                         : rem et exécuter
90  end
100 rem — programme-machine —
110 data 16188,-1,16188,-1,16188,-1
120 data 16188,-1,16188,-1,16188,15
130 data 16188,15,20046,-8196,0,14
140 data 20085
```

Comme on le voit, ce programme est bien plus court que celui qui a été désassemblé précédemment. Cela tient au tableau des paramètres du programme, qui doit être présent dans tout programme exécutable à partir du Desktop, pour que le GEMDOS puisse déterminer les adresses de début et de fin du programme. Par ailleurs, la fin d'un programme nécessite un appel spécial du GEMDOS, lorsque l'on rend à nouveau la main au Desktop ou au programme de commande.

Par contre, lors de l'appel avec une instruction CALL, aucun renfort de paramètres n'est nécessaire et une simple instruction RTS en fin du programme machine (RTS correspond à l'instruction-BASIC RETURN) suffit pour revenir au programme BASIC appelant.

Le programme machine est à nouveau lu dans des lignes de data et POKé dans la mémoire. La particularité de cet exemple de programme réside dans le fait que cette mémoire est susceptible d'être décaler à volonté. Pour cette raison, on peut affecter le programme machine à la chaîne de caractères A$, dont l'adresse en mémoire peut être quelconque. Cette méthode possède un grand avantage : on peut de cette façon insérer plusieurs routines machines dans un programme BASIC, sans avoir à se faire de soucis sur leurs positions en mémoire. L'utilisation de n'importe quelle zone mémoire fixe pourrait en effet engendrer des problèmes, si le GEM ou le GEMDOS utilise également cette zone de travail en détruisant ainsi le programme machine. L'appel d'un tel programme conduit alors au plantage de l'ordinateur.

Dans la variable A$ se trouve donc, après la fin de la boucle FOR NEXT, notre programme machine. La prochaine étape est la saisie du code correspondant au débit souhaité, une vitesse quelconque pouvant être choisie. Le mot 'quelconque' est cependant exagéré, mais les 16 valeurs de débit possibles suffisent largement et répondent à tous les besoins concernant la transmission série. On entre donc le code du débit souhaité, qui est compris entre 0 (19200 bauds) et 15 (50 bauds). Vous trouverez une table complète des débits autorisés dans le chapître 5.1 de ce livre.

La valeur saisie est maintenant introduite dans le programme, en la POKant à la place correspondante. Puis on appelle le programme machine proprement dit par un CALL B, où B est l'adresse de la variable A$ et donc du programme. Cette routine machine appelle alors, par l'instruction TRAP, le BIOS étendu et transmet les paramètres à l'interface série. Voici le programme machine, une fois désassemblé :

```
* redéfinition du débit *

MOVE.W #-1,-(SP)     * Caractère de synchro
MOVE.W #-1,-(SP)     * Transmitter Status Register
MOVE.W #-1,-(SP)     * Received Status Register
MOVE.W #-1,-(SP)     * USART Status Register
MOVE.W #-1,-(SP)     * XON/XOFF et RTS/CTS
MOVE.W #15,-(SP)     * Débit (ici : 50 bauds)
MOVE.W #15,-(SP)     * Commande
```

```
TRAP  #14          * Appel du XBIOS
ADD.L #14,SP       * Corriger la pile

RTS                * Retour au BASIC
```

Le 15, qui se trouve à la ligne 6 et qui détermine le débit, est le nombre que nous avons chargé avec l'instruction POKE. En outre, on peut, par la même instruction POKE, modifier le protocole de transmission qui est déterminé par XON/XOFF et RTS/CTS. L'adresse correspondante dans le programme ci-dessus est B+18. Un POKE B+18,n suffit donc pour mettre en place un nouveau protocole. La valeur -1 du paramètre de protocole signifie toujours que la procédure actuelle est conservée. Si par exemple vous avez choisi dans le point de menu 'Set RS232 config.' le protocole RTS/CTS, il sera encore valable avec le programme ci-dessus.

Cet exemple montre donc l'utilisation de l'instruction CALL pour appeler un programme machine. L'emploi répété de CALL B montre que la variable A$ contient toujours le programme et B son adresse. Il peut cependant arriver que ces adresses changent, lors de modifications effectuées à l'intérieur du programme. Il est donc conseillé de déterminer la valeur de la variable B avant chaque nouvel appel.

Nous allons à présent voir comment on peut transmettre des paramètres à un programme machine, sans avoir besoin de les POKEr. La syntaxe de l'instruction CALL permet d'écrire les valeurs à transmettre entre parenthèses, par exemple CALL X (1,2,3). Le programme machine doit ensuite, de son coté, collecter ces valeurs. Comme je manquais de documentation à ce sujet, j'ai dû écrire un exemple de programme qui lit ces valeurs et les renvoie, tout simplement. Ce programme n'est à dire vrai pas d'une utilité exeptionnelle, mais montre de façon claire le cheminement des paramètres. Les paramètres, qui sont entre parenthèses, sont déposés sur la pile, et leur nombre se trouve dans le registre D0.

Voici donc ce programme :

Partie BASIC :

```
10  rem ** test CALL **
20  a$=space$(100)          : rem faire de la place
30  a=peek(varptr(a$)+2)    : rem déterminer l'adresse
40  bload'testcall.prg',a   : rem charger le programme machine
90  end                     : rem fin de la première partie
100 call a (1,2,3)          : rem appel du programme-test
130 n=peek(8192)            : rem détermination du nombre
140 print n;' arguments :'
150 defdbl j                : rem déclaration d'un mot long
155 d=1                     : rem initialiser le compteur
160 for j=8208 to 8204+n*4 step 4
170 print d;'. : ';peek(j)  : rem affichage du paramètre
180 d=d+1                   : rem incrémenter le pointeur
190 next j                  : rem et ainsi de suite
```

Partie machine :

```
MOVE.W D0,$2000          * nombre de paramètres
MOVE.L 14(SP),$2010      * premier paramètre
MOVE.L 18(SP),$2014      * deuxième    "
MOVE.L 22(SP),$2018      * troisième   "
MOVE.L 26(SP),$201C      * quatrième   "
MOVE.L 30(SP),$2020      * cinquième   " , ce sera suffisant

RTS                      * et retour au BASIC
```

Cette possibilité de transmettre des paramètres est utilisable dans beaucoup de cas. Pour illustrer cela, nous allons maintenant examiner un exemple général permettant de manipuler des adresses, dont l'accès est en réalité seulement autorisé dans le mode privilégié du programme, le mode superviseur. Le programmeur BASIC peut normalement 'uniquement' travailler en mode utilisateur (user). De ce fait le microprocesseur ne lui autorise que des accès mémoires non privilégiés. L'avantage de cette restriction réside dans le fait qu'un programme normal ne peut pas accèder à la sphère 'intime' de l'ordinateur. Le système est ainsi capable de se protéger lui-même des manipulations interdites.

Cela est cependant gênant dans certains cas. Un accès BASIC, au moyen d'un POKE, à une zone mémoire protégée conduit immédiatement au plantage, de telle sorte que le programmeur se trouve à nouveau dans le Desktop et que ses données BASIC ont disparues. Cela est d'autant plus dommage que certaines de ces adresses privilégiées ouvrent des possibilités intéressantes. Le domaine des I/O en est un exemple : sa manipulation permet de programmer les composants spéciaux de l'ATARI ST.

```
10  rem ** accès privilégié **
20  a$=space$(36)                  : rem réserver de la place
30  b=peek(varptr(a$)+2)           : rem déterminer l'adresse
35  b=b+(b mod 2)                  : rem adresse paire
40  for i=0 to 34 step 2
50  read a                         : rem lire les données
60  poke b+i,a                     : rem et les écrire
80  next i
90  rem — Données du programme machine —
100 data -20356,2,26140,17063,16188
110 data 32,20033,23695,8815,14,12975
120 data 20,12032,16188,32,20033,23695,20085
```

Ce programme génère un programme machine qui permet un accès direct à (ou une modification d') une adresse privilégiée. Il est appelé par l'instruction CALL B (Adresse, Valeur), où Adresse est un mot long et Valeur un mot. Pour tester le bon fonctionnement du programme, on peut afficher un petit trait sur le bord supérieur de l'écran grâce à un CALL B (491520,21845) pour le ST avec 512Ko ou CALL B (1015808, 21845) pour la version avec 1Mo. Ceci est à vrai dire également possible avec un POKE, mais il s'agit ici uniquement d'un simple test...

Pour tester un accès privilégié, on peut par exemple modifier la fréquence de balayage vertical de l'image couleur, ce qui exige en effet un accès au Video-chip compétent. L'adresse correspondante est $FF820A, i.e. 16744970, et doit contenir pour un téléviseur PAL/SECAM (50 Herz) la valeur 2. Pour modifier cette valeur, il est nécessaire de multiplier la valeur désirée par 256, car c'est un mot (je rappelle qu'un mot = 2 octets = 16 bits) qui est mis en place. L'adresse ne doit cependant contenir qu'un seul octet, de telle sorte que l'octet de poids faible 'rate' sa cible.

En multipliant la valeur correspondant à la fréquence par la constante 256, $100 en héxadécimal, on affecte l'octet de poids faible à la bonne adresse.

Examinons à présent le programme machine, que nous avons 'affecter' à la variable A$.

Source File : TEST.S

```
1                       * Domaine privilégié *
2
3  000000 B07C0002      CMP.W     #2,D0       * 2 arguments ?
4  000004 661C          BNE       EXIT        * non => Exit
5
6  000006 42A7          CLR.L     - (SP)
7  000008 3F3C0020      MOVE.W    #$20,-(SP)
8  00000C 4E41          TRAP      #1          * Supervisor !
9  00000E 5C8F          ADD.L     #6,SP       * réparer la pile
10
11 000010 226F000E      MOVE.L    14(SP),A1   * chercher l'adresse
12 000014 32AF0012      MOVE.W    20(SP),(A1) * et affecter
13
14 000018 2F00          MOVE.L    D0,-(SP)
15 00001A 3F3C0020      MOVE.W    #$20,-(SP)
16 00001E 4E41          TRAP      #1          * User-State
17 000020 5C8F          ADD.L     #6,SP       * réparer la pile
18
19 000022 4E75      EXIT: RTS                 * et on revient !
```

Avant que le programme ne commence réellement, on teste si l'appel CALL a bien transmis 2 paramètres. Si ce n'est pas le cas, on branche directement à l'instruction RTS, où l'on rend la main au BASIC. Si tout est en ordre, l'instruction TRAP met le système, les arguments étant déposés sur la pile, en mode superviseur, où l'on travaille de manière privilégiée. Après le TRAP, il faut, comme toujours, réparer la pile pour la remettre à jour, en modifiant simplement la valeur du pointeur de pile.

Au pas suivant, le programme charge l'adresse transmise dans le registre A1. Puis il affecte à cette adresse, de manière indirecte, la valeur à écrire. C'est tout. Il suffit maintenant de remettre le système dans le mode utilisateur : c'est le rôle du TRAP suivant. Enfin, on se rebranche par l'instruction RTS au programme appelant, c'est-à-dire le programme BASIC.

Le format du listing du programme machine désassemblé est établi par l'assembleur du kit de développement, lorsque, après l'appel de l'assembleur, on entre la commande >TEST.ASM. Ce nom de programme arbitraire est alors attribué au fichier de sortie contenant le listing. Ce fichier contient encore quelques informations supplémentaires, comme le nombre d'erreurs et la liste des labels. Ces informations sont ici inutiles. Par contre, il est intéressant de noter la correspondance entre les nombres hexadécimaux générés et les valeurs décimales introduites dans les lignes de datas de notre programme BASIC.

7. PROGRAMMATION DE L'ATARI ST EN BASIC

Après toutes ces remarques sur l'intérieur et l'extérieur de l'ATARI ST, nous allons à présent nous intéresser à la programmation de l'ordinateur. Nous utiliserons, pour nos applications, le langage BASIC, car c'est le langage préféré des programmeurs. Par ailleurs, le BASIC possède l'avantage suivant : les programmes sont faciles à survoler et à comprendre, de telle sorte que les exemples qui vont suivre peuvent facilement être reproduits et insérés dans vos propres applications. Vous avez déjà rencontré quelques programmes dans ce livre qui sont écrits dans ce langage, et vous avez peut-être entrepris telle ou telle modification. De telles manipulations ne sont pas dangereuses en BASIC, car, dans le pire des cas, il apparait un message d'erreur.

Mais cela n'est plus aussi sûr avec des programmes contenant des PEEKs et des POKEs. En effet, en utilisant ces instructions, et en particulier l'instruction POKE, on entreprend d'accèder dans le domaine machine. Un seul mauvais POKE peut facilement tout 'planter'. Avant donc que vous n'expérimentiez ces instructions, vous devriez vous assurer que rien ne sera perdu dans le cas d'un eventuel plantage. Il est donc conseillé de sauvegarder les programmes avant de les exécuter, car une faute de frappe est vite arrivée...

Chargeons donc l'interprèteur BASIC et commençons la partie la plus intéressante quant à l'utilisation d'un ordinateur, la programmation elle-même.

Les programmes présentés dans les paragraphes suivants contiennent, comme l'indique le titre du livre, beaucoup d'instructions PEEK et POKE. Celui qui a déjà travaillé sur un ordinateur en BASIC ne verra pas de difficultés à utiliser ces instructions. PEEK lit le contenu d'une adresse et POKE le modifie.

Rien ne parait plus simple, mais il y a cependant un point dont il faut absolument tenir compte, quand on a devant soi un ordinateur 16 bits comme l'ATARI ST et que l'on travaille avec Personal-Basic. Le microprocesseur de cet ordinateur travaille en effet avec des octets (8 bits), des mots(16 bits) et des mots longs(32 bits). Le BASIC du ST, traite les mêmes entités.

C'est pour cette raison qu'on obtient souvent avec un PEEK une valeur immensément grande, qui n'entrerait certainement pas dans un octet.

Il est donc important de connaître ou de définir la taille des cellules mémoires employées, si l'on veut manipuler celles-ci avec POKE. Supposons que nous voulions mettre à zéro l'octet situé à l'adresse 8000. Si nous écrivons simplement POKE 8000,0, notre octet aura certes la valeur 0, mais l'octet suivant à l'adresse 8001 aussi. Cela provient tout simplement du fait que l'instruction POKE ne manipule normalement que des mots (16 bits) dans la mémoire, de telle sorte que l'accès à un octet seul n'est pas possible de cette manière.

Un autre problème apparaît quand on veut modifier la valeur d'un pointeur. Un pointeur est habituellement un mot long (32 bits), sa taille est donc de 4 octets. Il faudrait donc utiliser deux instructions POKE pour modifier entièrement un tel pointeur. Cette méthode a deux inconvénients. Premièrement, il faut diviser la valeur du pointeur que l'on veut charger en un mot de poids fort et un mot de poids faible, ce qui exige un calcul préalable supplémentaire dans le programme. Deuxièmement, le pointeur va avoir entre les deux POKEs une valeur non prévue. Cela peut avoir des effets assez désagréables, si le pointeur est utilisé dans une routine d'interruption juste avant le deuxième POKE. Le pointeur a alors une valeur imprévue et peut, suivant l'utilisation qu'en fait la routine d'interruption, conduire au plantage du système.

Les programmeurs de l'interprèteur BASIC ont réalisé qu'il y avait là un problème et ont donné la possibilité de définir la taille de la cellule mémoire à laquelle on veut accéder.

Pour différencier si l'on veut accéder par PEEK (ou POKE) à un mot ou à un mot long, on déclare la taille des variables représentant les adresses utilisées par cette instruction. Si la variable A de l'instruction X=PEEK(A) ou POKE A,X a été définie comme une variable double précision, alors on lit ou on écrit un mot long. Une telle déclaration est possible grâce à l'instruction DEFDBL A.

Si cette déclaration est omise ou si l'on déclare la variable comme une variable simple précision avec DEFSNG A, l'instruction PEEK ou POKE manipule alors uniquement des mots.

Il nous manque encore la possibilité de pouvoir indiquer qu'on lit ou écrit un octet. Pour cela, il faut une instruction supplémentaire que le BASIC met à notre disposition. Il s'agit de DEF SEG = 1 qui a deux effets. D'une part, on ne travaille dorénavant plus qu'avec des octets ; on peut revenir au traitement de mots initial avec DEF SEG = 0. D'autre part, on ne lit plus l'octet adressé lui-même, mais celui qui le suit. On doit donc diminuer de 1 l'adresse pour atteindre l'octet voulu. Cet adressage peu commode a cependant aussi un avantage. Comme l'offset, c'est-à-dire la différence entre l'adresse indiquée et l'adresse traitée, peut prendre une valeur entière quelconque, avec l'instruction DEF SEG = n (n est cet offset), on peut adresser de manière indexée. Cela veut dire que la séquence DEF SEG = n : X = PEEK(A) lit l'adresse A+n. On peut ainsi, en gardant l'adresse constante et en faisant varier l'offset, traiter toute une ligne de données, qui commencera cependant à l'adresse A+1. Un offset égal à 0 provoque le retour au traitement de mots.

Cette méthode consistant à déclarer la taille des données exige une programmation méticuleuse. J'ai souvent moi-même oublié de revenir au traitement de mots, de telle sorte que le programme tournait certes à la première exécution, mais se plantait à la deuxième. En effet, l'instruction RUN ne réinitialise pas la taille des variables ; il est donc indispensable de faire figurer explicitement cette déclaration dans le programme.

Si vous testez, au cours de vos essais, les programmes avec les instructions PEEK et POKE qui vont suivre, faites s'il vous plait attention à la définition exacte de la taille des données. Un programme tout a fait correct syntaxiquement ne peut fonctionner s'il traite des octets à la place de mots. Pour programmer en toute sécurité, il est conseillé de placer en première ligne d'un programme utilisant PEEK et POKE l'instruction DEF SEG = 0.

7.1 GRAPHISME

L'ATARI ST, avec sa haute résolution graphique et l'environnement GEM, est très bien équipé pour le graphisme. Le GEM gère à peu près tout ce qu'un programmeur peut imaginer du point de vue graphique.

Tirer des traits, tracer des cercles et des ellipses, dessiner des rectangles avec des coins arrondis ou non, et même remplir des formes quelconques avec un motif également quelconque sont des processus si simples avec le ST que le programmeur d'ordinateur ne possédant pas le GEM n'en croit pas ses yeux. Commençons donc tout de suite par dessiner un cercle plein.

Le programmeur susdit doit maintenant penser à un programme plein de fonctions sinus et cosinus. Mais nous n'avons pas besoin de cela. L'instruction PCIRCLE X,Y,R effectue immédiatement et rapidement ce tracé. Cependant, cette instruction possède une restriction : elle ne peut dessiner le cercle qu'à l'intérieur de la fenêtre OUTPUT de l'écran BASIC.

Cette restriction n'est plus valable si vous utilisez directement le GEM pour cette fonction graphique. On peut alors effectuer n'importe quel tracé à n'importe quelle place de l'écran. De plus, cette méthode directe offre plus de possibilités de formes que le BASIC.

7.1.1 Cercles, Ellipses et Rectangles

Pour tracer un cercle dans le coin supérieur gauche de l'écran, nous devons faire un appel VDISYS, comme nous l'avons déjà vu dans le chapitre sur le GEM. Le programme suivant trace un cercle et le remplit :

```
10  rem ** dessin d'un cercle plein **
20  color 1,0,1,1,1          : rem couleur du remplissage
30  poke contrl,11           : rem tracer un cercle
40  poke contrl+2,3          : rem nombre de paramètres
50  poke contrl+6,0
60  poke contrl+10,4         : rem ID de la fonction
70  poke ptsin,100           : rem abscisse X du centre
80  poke ptsin+2,100         : rem ordonnée Y du centre
90  poke ptsin+4,0           : rem Dummy
100 poke ptsin+6,0           : rem Dummy
110 poke ptsin+8,50          : rem rayon
120 poke ptsin+10,0          : rem Dummy
130 vdisys 0                 : rem et exécution
```

La fonction 11, dont le numéro est transmis au vecteur CONTRL(0) à la ligne 30, ne trace pas uniquement des cercles. Elle permet en effet d'afficher 10 formes graphiques différentes. L'information qui indique de quelle forme il s'agit est contenu dans le code ID propre à chaque forme, qui est affecté à CONTRL(5). La correspondance entre le code ID et la figure effectuée est la suivante :

ID	Figure
1	rectangle plein
2	secteur de cercle
3	secteur de cercle plein
4	cercle plein
5	ellipse
6	secteur d'ellipse
7	secteur d'ellipse plein
8	rectangle avec coins arrondis
9	rectangle plein avec coins arrondis
10	texte justifié

La première de ces formes, le rectangle plein, est très bien adaptée pour la réalisation de diagrammes rectangles. On peut en effet représenter des rectangles de largeur quelconque avec un attribut de remplissage quelconque, ce qui offre pour de tels diagrammes une grande diversité de possibilités. Un tel rectangle peut être réalisé par ce programme :

```
10  rem ** rectangle plein **
20  color 1,2,2,9,2              : rem attribut de remplissage
30  poke contrl,11               : rem remplir un rectangle
40  poke contrl+2,2              : rem nombre de paramètres
50  poke contrl+6,0
60  poke contrl+10,1             : rem ID de la fonction
70  poke ptsin,50                : rem abscisse de départ
80  poke ptsin+2,50              : rem ordonnée de départ
90  poke ptsin+4,100             : rem abscisse finale
100 poke ptsin+6,200             : rem ordonnée finale
110 vdisys 0                     : rem et exécution
```

Ici, les coordonnées de départ sont celles du coin supérieur gauche, les coordonnées finales étant celles du coin inférieur droit. La signification de l'attribut de remplissage sera expliquée plus loin.

La fonction suivante, qui dessine un secteur de cercle, correspond à l'instruction-BASIC CIRCLE X,Y,R,D,F et affiche à l'écran un secteur de cercle non rempli. Pour cela, il faut définir deux paramètres supplémentaires : l'angle initial et l'angle final du secteur. Cet angle est mesuré en 1/10 de degré .L'angle 0 correspond au point situé à droite du centre. Un quart de cercle dans le premier cadran, c'est-à-dire le cadran en haut à droite du centre, se trouve donc compris entre 0 et 900 (90 degré). La même chose est valable pour l'instruction CIRCLE.

Un exemple :

```
10  rem ** secteur de cercle **
20  color 1,0,2,1,1          : rem attribut de ligne
30  poke contrl,11           : rem dessiner un secteur
40  poke contrl+2,4          : rem nombre de paramètres
50  poke contrl+6,2
60  poke contrl+10,2         : rem ID de la fonction
70  poke ptsin,150           : rem abscisse du centre
80  poke ptsin+2,50          : rem ordonnée du centre
90  poke ptsin+4,0           : rem Dummy
100 poke ptsin+6,0
110 poke ptsin+8,0
120 poke ptsin+10,0
130 poke ptsin+12,30         : rem rayon
140 poke intin,0             : rem angle initial
150 poke intin+2,900         : rem angle final
160 vdisys 0                 : rem et exécution
```

Le même programme peut aussi dessiner un secteur rempli. Il suffit pour cela de remplacer le code ID par 3 et de corriger l'attribut de remplissage dans la ligne 20. Pour un cercle entier il ne faut naturellement pas définir les angles 0 et 360 degrés, mais utiliser la fonction 4. Celle-ci a d'ailleurs déjà été illustrée dans notre premier exemple.

Examinons à présent les 'cercles déformés', à savoir les ellipses. On dispose pour celles-ci des fonctions 5, 6 et 7. Prenons par exemple la fonction 6, qui dessine un arc d'ellipse, dont les extrémités sont reliées au centre.

```
10  rem ** secteur d'ellipse **
20  color 1,0,1,1,1              : rem attribut de ligne
30  poke contrl,11               : rem dessiner un secteur d'ellipse
40  poke contrl+2,2              : rem nombre de paramètres
50  poke contrl+6,2
60  poke contrl+10,6             : rem ID
70  poke ptsin,200               : rem abscisse du centre
80  poke ptsin+2,100             : rem ordonnée du centre
90  poke ptsin+4,100             : rem rayon suivant X
100 poke ptsin+6,50              : rem rayon suivant Y
110 poke intin,3200              : rem angle initial
120 poke intin+2,1200            : rem angle final
130 vdisys 0                     : rem et exécution
```

Pour la fonction 7, qui correspond au secteur d'ellipse plein, seuls changent la valeur de ID et l'attribut de remplissage, qui peut d'ailleurs être défini avec l'instruction COLOR. Par contre, la fonction 5 ne nécessite pas d'angles initial et final, car c'est en effet une ellipse complète qui est dessinée. Les lignes 110 et 120 disparaissent donc simplement, tandis que les lignes 50 et 60 sont modifiées en affectant 0 à CONTRL+6 et en remplaçant le code ID par 6.

Puis vient une forme, qui n'est pas réalisable de manière simple par un programme BASIC (l'interprèteur BASIC ne possède pas d'instruction correspondante). Il s'agit du rectangle dont les coins sont arrondis. On dispose pour réaliser cette figure des codes ID 8 et 9, dont la programmation est quasiment la même pour les deux. La seule différence est que pour la fonction dont l'ID est 9, le rectangle est rempli, alors qu'il ne l'est pas pour la fonction 8. Il suffit donc de modifier le code ID et les attributs de remplissage/ligne de l'instruction COLOR.

```
10  rem ** rectangle avec coins arrondis **
20  color 1,2,2,3,4          : rem attribut de remplissage
30  poke contrl,11           : rem dessiner un rectangle
40  poke contrl+2,4          : rem nombre de paramètres
50  poke contrl+6,0
60  poke contrl+10,9         : rem ID
70  poke ptsin,50            : rem abscisse initiale
80  poke ptsin+2,50          : rem ordonnée initiale
90  poke ptsin+4,200         : rem abscisse finale
100 poke ptsin+6,200         : rem ordonnée finale
110 vdisys 0                 : rem et exécution
```

Jusqu'ici, nous n'avons étudié que les fonctions graphiques de base, que gère avec le GEM-VDI. Mais pour rendre les graphiques compréhensibles, il faut pouvoir y ajouter des inscriptions. Cela entre également dans le cadre du VDI. Le VDI peut afficher, de diverses façons, du texte sur l'écran graphique. Nous en arrive déjà au thème suivant :

7.1.2 Réalisation de texte graphique

Le dernier des appels graphiques du VDI est la fonction dont l'ID est 10. Cette fonction permet d'afficher de diverses façons un texte formatté à l'écran - naturellement aussi à une place quelconque. Cette fonction possède une flopée de paramètres. Il y a d'abord bien sûr la position à partir de laquelle le texte doit être écrit. Ces coordonnées indiquent le point supérieur gauche du premier caractère du texte. Puis on transmet la longueur totale de la place qu'on souhaite voir occuper par le texte. Si celle-ci est plus grande que la chaîne de caractères, on justifie le texte en insérant uniformément des blancs (caractère espace). Cet allongement peut être pris en compte ou non, de telle sorte que l'on peut choisir si ce sont les espaces vides ou bien les caractères eux-mêmes qui s'étirent.

Enfin, on transmet le texte lui-même. Celui-ci doit être affecté caractère par caractère au vecteur INTIN. L'opération est donc un peu plus longue que pour les figures précédentes. Mais ce n'est pas aussi compliqué que ça en a l'air.

Voici un programme réalisant cette application :

```
10  rem ** texte graphique - Demo VDISYS **
20  texte$= 'texte-essai'
30  poke contrl,11          : rem code instruction
40  poke contrl+2,3         : rem nombre de paramètres
50  poke contrl+6,len(texte$)+2
60  poke contrl+10,10       : rem indice ID de la fonction
70  poke intin,1            : rem étirement des mots (0=off)
80  poke intin+2,1          : rem étirement des caractères
                              (0=off)
90  poke ptsin,50           : rem abscisse
100 poke ptsin+2,100        : rem ordonnée
110 poke ptsin+4,50         : rem longueur du texte suivant X
120 for i=1 to len(texte$)  : rem caractères ASCII
130 poke intin+2+i*2,asc(mid$(texte$,i,1))   : rem affecter
140 next i
150 vdisys 0                : rem et exécution
```

Comme nous parlons justement de texte, nous devrions tout de suite examiner quelles sont les possibilités de formes de textes que propose le GEM. Dans le paragraphe 4.2.4, nous avons déjà appris comment choisir l'un des jeux de caractères de l'ATARI ST. Cependant, il est quelquefois souhaitable de pouvoir écrire des lettres ou des chiffres de différentes tailles et dans des directions quelconques. Ici aussi le GEM a tout prévu. Voyons, à ce sujet, un programme qui affiche de façon presque magique des immenses caractères au milieu de l'écran (dont la taille n'est cependant pas encore maximale...).

```
10  rem ** modification de la taille des caractères **
20  poke contrl,12              : rem code instruction
30  poke contrl+2,1             : rem nombre de paramètres
40  poke contrl+6,0
50  poke ptsin,0                : rem Dummy
60  poke ptsin+2,30             : rem hauteur des caractères
70  vdisys 0                    : rem et exécution
80  texte$= 'Quel texte !'
90  poke contrl,8               : rem texte
100 poke contrl+2,2
110 poke contrl+6,len(texte$)
120 poke ptsin,100              : rem abscisse
```

```
130 poke ptsin+2,100                    : rem ordonnée
140 for i=1 to len(texte$)
150 poke intin+2*i-2,asc(mid$(texte$,i,1))
160 next i
170 vdisys 0                            : rem écrire le texte
180 poke contrl,12 : poke contrl+2,1
190 poke contrl+6,0 : poke ptsin+2,13
200 vdisys 0                            : rem et normaliser
```

Ici nous avons utilisé simultanément deux commandes VDI, dont les numéros sont 8 et 12. La commande 12 détermine la hauteur des caractères à afficher. Cette hauteur est affectée à PTSIN(1). C'est le seul paramètre de la fonction. Le 0 dans PTSIN(0) n'a pas de fonction, mais doit être transmis pour des raisons de sécurité.

La partie du programme qui suit appelle la fonction 8, qui écrit le texte aux coordonnées introduites X et Y de l'écran. Ici, une simple instruction PRINT aurait aussi été possible ; mais cette dernière ne couvre que la fenêtre OUTPUT.

Enfin, après avoir afficher le texte, on remet en place le format standart. Ceci doit obligatoirement être fait, car l'éditeur de l'écran ne sait pas trop quoi faire avec ce texte gigantesque, de telle sorte que les saisies suivantes entrées dans la fenêtre commande ou la fenêtre OUTPUT n'auraient plus aucun effet.

Avec cette fonction, on peut déjà obtenir quelques effets intéressants. Mais comment faire, si, dans un dessin, l'on veut afficher une inscription le long d'une ligne verticale ? Encore une fois, le VDI permet de le réaliser. Avec la fonction 13 on peut donner à la ligne de base, le long de laquelle est écrit le texte, un angle de 90 degrés. Un angle 0 correspond à l'affichage normal. L'angle 900 (en 1/10 de degré) permet d'écrire un texte de bas en haut, 1800 le met à l'envers et 2700 a pour effet d'écrire le texte de façon lisible de haut en bas. Malheureusement, l'écran monochrome de l'ATARI ST n'autorise pas d'angles intermédiaires, qui sont reservés aux tables traçantes.

Voici donc un programme qui affiche un texte verticalement :

```
10  rem ** modifier la direction d'écriture **
20  poke contrl,13                  : rem code instruction
30  poke contrl+2,0                 : rem nombre de paramètres
40  poke contrl+6,1
50  poke intin,900                  : rem 90 degrés
70  vdisys 0                        : rem et exécution
80  texte$= 'Texte vertical !'
90  poke contrl,8                   : rem texte
100 poke contrl+2,2
110 poke contrl+6,len(texte$)
120 poke ptsin,100
130 poke ptsin,300
140 for i=1 to len(texte$)          : rem transmettre le texte
150 poke intin+2*i-2,asc(mid$(texte$,i,1))
160 next i
170 vdisys 0                        : rem écrire le texte
180 poke contrl,13 : poke contrl+2,0
190 poke contrl+6,1 : poke intin,0
200 vdisys 0                        : rem et exécution
```

Ici encore, la fonction 8, qui affiche un texte, a été utilisée. L'instruction normale PRINT ne peut pas être employée. En effet, PRINT n'écrirait que des moitiés de caractère couchés, ce qui n'est pas précisément très lisible. Comme dans l'exemple précédent, on remet ensuite immédiatement l'écriture standard - pour les même raisons que précédemment.

Ce sera tout, en ce qui concerne l'affichage de texte. Mais le tracé de cercles et l'ajout de texte ne suffisent pas pour obtenir un bon graphisme. Il faut de plus pouvoir tirer des droites - si possible de différents types. De même, la représentation de descripteurs et de flèches est indispensable dans un dessin technique. Cela ne pose naturellement aucun problème au VDI.

7.1.3 Lignes droites et marques

Tracer une droite dans la fenêtre OUTPUT est habituellement la fonction de l'instruction LINEF. On introduit ici les positions de départ et d'arrivée de la ligne. La fonction correspondante du VDI a pour numéro 6 et porte le nom de Polyline. Polyline désigne le tracé de plusieurs droites jointives. On transmet à la fonction les coordonnées de plusieurs points, qui sont ensuite reliés suivant l'ordre dans lequel ils ont été entrés. On peut ainsi facilement dessiner des formes simples, comme des cadres, car un seul appel suffit.

Voici encore un exemple :

```
10  rem ** ligne brisée **
30  poke contrl,6              : rem code polyline
40  poke contrl+2,4            : rem nombre de points
50  poke contrl+6,0
60  poke ptsin,50              : rem X 1
70  poke ptsin+2,50            : rem Y 1
80  poke ptsin+4,150           : rem X 2
90  poke ptsin+6,100           : rem Y 2
100 poke ptsin+8,160           : rem X 3
110 poke ptsin+10,250          : rem Y 3
120 poke ptsin+12,50           : rem X 4
130 poke ptsin+14,50           : rem Y 4
140 vdisys 0                   : rem et exécution
```

Cet appel trace un triangle à l'écran. Pour ce faire, il a naturellement fallu introduire le point de départ (ou, si l'on veut, le point d'arrivée) deux fois. On affecte à CONTRL(1) le nombre de points transmis. Cette fonction est analogue à la fonction dont le code instruction est 9. Celle-ci, en plus du tracé du polygone, remplit ce dernier avec le motif souhaité. Pour utiliser cette fonction, il suffit de rajouter l'instruction COLOR, qui détermine l'attribut de remplissage.

Revenons cependant aux lignes droites. Il serait intéressant de pouvoir tracer des lignes pointillées. Nous allons donc en faire part au VDI. A cet effet est prévue la fonction 15, pour laquelle on peut choisir jusqu'à 7 types différents de trait.

Voici la signification de chaque type de trait pouvant être transmis à INTIN(0) :

Type	Trait	
1	————————	continu
2	——— ———	interrompu
3	— — — —	pointillé
4	—— — —— —	mixte (trait-point)
5	—— ——	pointillé long
6	—— — — —— — —	trait-point-point
7	redéfinissable soi-même	

Le type 7 peut être redéfini. Une ligne est entièrement définie par un mot de 16 bits qui constitue le masque du motif. Le trait continu a donc pour masque 1111111111111111 = $FFFF = 65535. Cette valeur est alors transmise à la fonction 113 et détermine la forme de chaque ligne qui est tracée avec le numéro de type 7.

Voici un programme qui montre à la fois la redéfinition du motif et le choix du type de trait :

```
10  rem ** set line type **
20  clearw 2                : rem effacer la fenêtre OUTPUT
30  poke contrl,15          : rem choix du type de trait
40  poke contrl+2,0         : rem nombre de paramètres
50  poke contrl+6,1
60  poke intin,7            : rem type redéfinissable
70  vdisys 0                : rem et exécution
80  poke contrl,113         : rem définir le trait
90  poke contrl+2,0         : rem nombre de paramètres
100 poke contrl+6,1
110 poke intin,65365        : rem masque du motif
120 vdisys 0                : rem et execution
130 linef 0,20,200,20       : rem ligne d'essai
```

On peut également représenter les lignes avec une épaisseur variable. Il faut cependant noter que seules les lignes de type 1, c'est-à-dire les lignes continues, peuvent avoir différentes épaisseurs. S'il s'agit d'un autre type, le choix de l'épaisseur est tout bonnement ignoré. L'épaisseur du trait peut être choisie entre 1 et 15, en appelant la fonction 16.

Un exemple :

```
10  rem ** mise en place de l'épaisseur des traits **
20  clearw 2                : rem effacer la fenêtre OUTPUT
30  poke contrl,16          : rem mise en place de l'épaisseur
                              du trait
40  poke contrl+2,2         : rem nombre de paramètres
50  poke contrl+6,0
60  poke ptsin,15           : rem épaisseur du trait
70  vdisys 0                : rem et exécution
100 linef 0,30,250,100      : rem ligne d'essai
```

On peut encore redéfinir les extrémités de ces lignes.

Pour cela, on a le choix entre 3 formes possibles :

0 angle normal
1 flèche
2 arrondi

Ces formes sont mises en places avec la fonction 108. Les deux extrémités de la droite peuvent être définies séparément. Pour dessiner une flèche allant de la position X1/Y1 à la position X2/Y2 et arrondir son origine, nous pouvons utiliser le programme suivant :

```
10 rem ** dessin d'une flèche **
20 poke contrl,108          : rem redéfinition des extrémités
30 poke contrl+2,0
40 poke contrl+6,2
50 poke intin,2             : rem arrondir l'origine de la
                              droite
60 poke intin+2,1           : rem extrémité en forme de flèche
70 vdisys 0                 : rem mettre en place
```

Puis, il faut encore exécuter le dessin lui-même comme dans l'exemple précédent. L'arrondi de l'extremité d'une ligne n'a de sens que pour des traits épais. Les flèches en revanche sont représentées pour toutes les épaisseurs de trait.

7.1.4 Remplissage de surfaces quelconques

Lorsqu'on a peu à peu créé et dessiné une forme, il est intéressant (et esthétique) de pouvoir la remplir. Pour les graphiques situés à l'intérieur de la fenêtre OUTPUT, cela est possible avec l'instruction FILL X,Y du BASIC. Cette instruction remplit tout l'intérieur de la surface contenant le point X,Y et délimitée par une courbe enveloppe quelconque. Le motif de remplissage et sa couleur sont définis auparavant avec l'instruction COLOR. Cette dernière instruction possède 5 paramètres, dont la signification est la suivante :

Paramètres de l'instruction COLOR A,B,C,D,E

A Couleur de l'encre pour les textes
B Couleur de remplissage pour l'instruction FILL
C Couleur des lignes pour les graphiques
D Indice du motif de remplissage
E Type de motif de remplissage

L'indice et le type déterminent le motif de remplissage. En particulier, le type indique de quelle manière est effectué le remplissage.

Type	**Signification**
0	La surface n'est pas remplie
1	La surface est entièrement remplie
2	Remplissage de points
3	Remplissage de hachures
4	Remplissage avec un motif redéfinissable

L'indice joue uniquement un rôle pour les types 2 et 3 et précise le motif de remplissage.

7.1.5 Redéfinition du motif de remplissage

Le type 4 est donc très intéressant. Il est en effet possible de remplir une surface avec un motif quelconque. On peut par exemple choisir le symbole d'une société pour un en-tête de lettre, ou remplir l'écran avec des petits coeurs. L'imagination est uniquement limitée par les dimensions 16x16 du motif lui-même. Nous allons, dans notre exemple, reprendre comme motif de remplissage le bonhomme que nous avions créé lors de la redéfinition de la forme de la souris - car, au fond, c'était pas mal du tout.

```
10  rem ** définition du motif de remplissage **
20  poke contrl,112              : rem code instruction
30  poke contrl+6,16             : rem nombre de paramètres
80  for i=0 to 15 : read x$      : rem motif de remplissage
82  x=0 : for j=1 to 16          : rem transformation
84  x= x-(mid$(x$,j,1)<>' ')*2^(16-j)
86  next j
90  poke intin+i*2,x             : rem mettre en place le motif
100 next i
110 vdisys 0                     : rem et exécution
112 color 1,1,1,1,4              : rem choix du motif
114 pcircle 80,80,70             : rem et démonstration
120 end
130 rem +++ données du motif +++
140 data '                '
150 data '     ***        '
160 data ' **  * * *      '
170 data '  **  ***       '
180 data '   **  *        '
190 data '    ********    '
200 data '     ***** **   '
210 data '     *****  *   '
220 data '     ***** **   '
230 data '     *****      '
240 data '     ** **      '
250 data '     ** **      '
260 data '     ** **      '
270 data '     ** **      '
280 data '   **** ****     '
290 data '                '
```

Pour pouvoir à présent afficher ce motif ou un autre motif prédéfini à une position quelconque de l'écran, un appel du VDI est à nouveau indispensable. Il s'agit cette fois de la fonction 103, qui est aussi appelée par l'instruction-BASIC FILL. L'utilisation est donc exactement la même que pour un FILL, en ce qui concerne l'attribut et le déroulement du processus de remplissage.

Voici comment on peut effectuer cet appel dans un programme :

```
10 rem ** Demo - remplissage **
15 color 1,1,1,3,2              : rem attribut de remplissage
20 poke contrl,103              : rem commande de remplissage
30 poke contrl+2,2              : rem nombre de paramètres
40 poke contrl+6,1
50 poke intin,1                 : rem couleur de remplissage
60 poke ptsin,100               : rem abscisse
70 poke ptsin+2,200             : rem ordonnée
80 vdisys 0                     : rem et exécution
```

Vous pouvez du reste également hachurer ou pointiller la bande contenant le menu du GEM, si vous voulez ajouter cette touche personnelle.

7.1.6 Affichage de marques à l'écran

Dans certains programmes, il est avantageux de placer quelques repères sur l'écran. On peut certes dessiner des petits cercles avec l'instruction CIRCLE ou travailler avec l'ordre graphique du VDI qui correspond. Cela n'est cependant pas nécessaire, car le VDI possède des instructions de marquage. La forme de ces marques peut être choisie parmi tout un ensemble de motifs.

Ces types prédéfinis sont :

Type	Forme
1	Point
2	Caractère plus : +
3	Etoile : *
4	Carré
5	Croix : X
6	Diamant

Toutes les valeurs supérieures donnent une étoile. Ces symboles, et même le point, peuvent être aggrandis, de telle sorte qu'ils peuvent prendre exactement la taille souhaitée. Ainsi, avec le carré adéquat, on peut par exemple entourer un caractère.

Le choix du type, la déclaration de la taille, le choix de la couleur des marques et l'affichage des symboles sont autant de fonctions nécessitant chacune l'appel d'une fonction particulière. Cela augmente donc de façon sensible le nombre d'instructions, mais offre une grande souplesse d'utilisation. Beaucoup de programmes utilisent cette méthode de marquage pour indiquer d'une croix les options de menus choisies.

Examinons donc un exemple de programme qui affiche quelques marques après avoir effectué toutes les mises au point :

```
10  rem ** Poly-marker **
20  poke contrl,18          : rem type de marque
30  poke contrl+2,0         : rem nombre de paramètres
40  poke contrl+6,1
50  poke intin,5            : rem type
60  vdisys 0                : rem mettre en place

70  poke contrl,19          : rem taille
80  poke contrl+2,1
90  poke contrl+6,0
100 poke ptsin,0            : rem Dummy
110 poke ptsin+2,30         : rem hauteur des marques
120 vdisys 0                : rem mettre en place

130 poke contrl,20          : rem mettre en place la couleur
140 poke contrl+2,0
150 poke contrl+4,1
160 poke intin,2            : rem numéro de couleur
170 vdisys 0                : rem mettre en place

180 poke contrl,7           : rem polymarker
190 poke contrl+2,2         : rem nombre de points
200 poke contrl+6,0
210 poke ptsin,50           : rem X 1
220 poke ptsin+2,50         : rem Y 1
```

```
230 poke ptsin+4,150          : rem X 2
240 poke ptsin+6,100          : rem Y 2
250 vdisys 0                  : rem et dessiner
```

L'appel de la fonction 7, affichage des marques, dessine d'un coup tous les symbôles dont les coordonnées ont été transmises. Ces coordonnées Xn/Yn peuvent, comme toujours, représenter un point quelconque de l'écran. Notons également que tous les appels ultérieurs de la fonction utiliseront à nouveau les attributs qui ont été mis en place ici.

Après avoir affiché, avec cette fonction et le type 1, quelques points à l'écran, il arrive souvent qu'il faille déterminer si un point de l'écran est 'allumé' ou non. Cela est également possible grâce au VDI.

7.1.7 Test de la couleur d'un point de l'écran

La fonction 105 du VDI offre la possibilité de déterminer la couleur d'un point quelconque de l'écran. Si cette couleur est celle du fond, il apparait en outre un message indiquant l'état 'non allumé'. Ces informations sont obtenues, après l'appel de la fonction, dans INTOUT(0) et INTOUT(1). Elles sont très utiles, si l'on veut savoir, dans un jeu par exemple, si un objet a atteint une ligne frontière. La fonction 105 peut aussi servir à copier des blocs images. Voici un programme permettant d'interroger l'état d'un point :

```
10 rem   ** test d'un point **
20 poke contrl,105            : rem code instruction
30 poke contrl+2,2            : rem nombre de paramètres
40 poke ptsin,x               : rem x: abscisse du point
50 poke ptsin+2,y             : rem y: ordonnée du point
70 vdisys 0                   : rem et tester
80 etat = peek(intout)        : rem 1 = allumé / 0 = éteint
90 couleur = peek(intout+2)   : rem 0/1 pour l'écran monochrome
```

7.1.8 Mélange de couleurs

Avec le panneau de contrôle, on peut définir les 16 couleurs que l'ATARI ST peut représenter simultanément sur un moniteur couleur. Ces couleurs sont utilisées dans un dessin en choisissant le numéro courant de 0 à 15 grâce à l'instruction COLOR. Cependant, il est parfois nécessaire de faire varier ces couleurs pendant l'exécution d'un programme. Dans ce cas, il est impossible d'utiliser le panneau de contrôle. Le BASIC n'offre aucune instruction pour la manipulation des registres couleurs eux-mêmes. Il faut donc à nouveau avoir recours au VDI.

La fonction 14 du VDI permet de redéfinir un registre couleur quelconque. Une couleur est en fait un mélange des 3 couleurs de base. Elle est entièrement déterminée par les intensités respectives de ces 3 couleurs de base. L'échelle des intensités va de 0 à 1000. N'importe quelle couleur peut ainsi être créée.

Les trois valeurs d'intensité sont transmises simultanément à la fonction et immédiatement mis en place par l'appel du VDI. Les graphiques et les dessins qui ont été réalisés avec les couleurs redéfinies prennent alors immédiatement le nouveau ton de couleur. En effet, même en ayant recours à des astuces, il n'est pas possible de tirer du ST plus de 16 couleurs !

Voici un exemple de programme donnant un ton brun à la couleur numéro 2 :

```
10 rem  ** redéfinition de couleur **
20 poke contrl,14          : rem numéro de fonction
30 poke contrl+2,0
40 poke contrl+6,4
50 poke intin,2            : rem numéro de couleur
60 poke intin+2,600        : rem part du rouge
70 poke intin+4,400        : rem part du vert
80 poke intin+6,200        : rem part du bleu
90 vdisys 0                : rem et mise en place
```

Pour réaliser la couleur souhaitée, en faisant varier les taux du rouge, du vert et du bleu, il vous faut certes un peu essayer à droite et à gauche. Mais cela aussi est intéressant, car on découvre ainsi la diversité des couleurs de l'ATARI ST.

7.2 BRUIT ET MUSIQUE

L'ATARI ST dispose, pour la génération de sons et de bruits, de trois générateurs de sons, que l'on peut mixer à souhait. De plus, chacun de ces trois canaux peut être utilisé comme générateur de bruit, en programmant convenablement le Sound-Chip. Enfin, ce Chip possède un générateur d'enveloppe incorporé, avec lequel on peut obtenir assez facilement des effets de bruitage intéressants.

Ces sons sont obtenus en combinant les instructions SOUND et WAVE. Ces instructions offre une multitude de possibilités et permettent de produire un bruit très réaliste et très proche de celui qu'on se représente.

La façon la plus simple, pour générer un son, est d'appuyer sur une touche. Le 'beep' qui s'en suit est en effet également un son produit par le Sound-Chip. Cela explique pourquoi l'enfoncement d'une touche interrompt un son programmé. En effet, lorsqu'un beep doit être généré, le système d'exploitation reprogramme entièrement le Sound-Chip. De ce fait, une partie au moins des sons mis au point par les programmeurs ne sont souvent pas pris en compte.

Le beep du clavier et la cloche qui retentit lors d'un message d'erreur (si cela est mis en place dans le champ de contrôle), sont générés dans les interruptions. Pour cela, le système d'exploitation a uniquement besoin de pointer sur une table, qui contient les données du programme du Sound-Chip. Le programme d'interruption, qui tourne presque en permance, teste à chaque passage si ce pointeur est à 0. Si cette valeur est différente de 0, elle est interprètée comme un véritable pointeur. La routine d'interruption utilise alors les valeurs du tableau qui est pointé, et programme grâce à celles-ci le son à générer. Si elle tombe dans le tableau sur la valeur $FF, son travail s'achève et le système d'exploitation reprend le contrôle.

Avant de tester tout cela, écoutons tout d'abord un de ces sons préprogrammés. Pour cela, plaçons le pointeur, qui se trouve à l'adresse $A86, sur la table correspondante du système d'exploitation.

Pour un beep du clavier, nous écrivons :

```
10 DEFDBL A
20 A = 3652
30 POKE A,36642
```

Un son de cloche est obtenu par :

```
30 POKE A,36612
```

Nous pouvons naturellement créer nous-même une telle table, afin de pouvoir générer un bruit ou toute une série de sons avec un seul POKE. Avant de passer à la pratique, nous devrions examiner de plus près la programmation du Sound-Chip.

Le transfert des paramètres au Sound-Chip est effectué dans le domaine des I/O de la mémoire. Le problème est donc toujours le même : l'accès à cette zone mémoire est privilégié et conduit avec un simple POKE au plantage pur et simple. Mais nous connaissons déjà une recette 'brevetée', à savoir le contournement du problème grâce à un petit programme machine, qui autorise cet accès en mettant le système dans le mode superviseur.

Nous avons déjà écrit un tel programme dans le chapître 6.4.1. On peut, après l'exécution de celui-ci, entreprendre un accès privilégié grâce à un CALL.

Chargeons donc le programme de la disquette et ajoutons y les lignes suivantes:

```
200 read r,w                        : rem registre et valeur
205 if r=-1 then end                : rem test de fin
210 call b(16746496,r*256)          : rem choisir le registre
215 call b(16746498,w*256)          : rem et affecter la valeur
220 goto 200                        : rem continuer...
249 rem             utilisation des canaux A et B
250 data 7,252
259 rem             intensité du canal A
260 data 8,12
269 rem             Canal B sur courbe enveloppe, fréquence
270 data 9,16,3,2
279 rem             fréquence de l'enveloppe du son
280 data 13,10
299 rem             Fin !
300 data -1,0
```

Ce programme effectue le mixage de deux fréquences, dont l'une varie en fonction d'une courbe enveloppe. Les données à partir de la ligne 250 ont la signification suivante :

Il s'agit à chaque fois d'un couple de données: La première indique le numéro de registre utilisé dans le Sound-Chip et la deuxième est la valeur qui doit être écrite dans ce registre. Nous avons ici la possibilité de faire varier ces valeurs pour générer le son souhaité. Les registres et leurs contenus ont des significations différentes que nous allons maintenant examiner :

Nr - registre	**Signification**
0 et 1	Durée de la période du canal A. En tout, 12 bits de ce mot sont utilisés.
2 et 3	Idem avec le canal B
4 et 5	Idem avec le canal C
6	Période du générateur de bruit. Les 5 bits de poids faible sont utilisés
7	Registre de configuration. Chaque bit a un rôle particulier : Bit 0 : canal A on/off (0 = on, 1 = off) Bit 1 : canal B on/off (0 = on, 1 = off) Bit 2 : canal C on/off (0 = on, 1 = off) Bit 3 : bruits sur canal A (0 = oui, 1 = non) Bit 4 : bruits sur canal B (0 = oui, 1 = non) Bit 5 : bruits sur canal C (0 = oui, 1 = non) Bit 6 : sens de la transmission au port A (0 = in, 1 = out) Bit 7 : sens de la transmission au port B (0 = in, 1 = out)
8	Intensité sonore du canal A. Les 4 bits de poids faible sont utilisés. Si le bit 4 est à 1, ce canal est déterminé par la courbe enveloppe et les 4 bits de poids faible sont ignorés.

9	Idem avec le canal B
10	Idem avec le canal C
11 et 12	Durée de la période de la courbe enveloppe (11:LO, 12:HI). Tous les 16 bits sont nécessaires.
13	Forme de la courbe enveloppe (Bits 0 à 3). Ces formes de courbes sont difficiles à décrire ; il faut donc les essayer !
14	Registre de donnée relié au port A
15	Registre de donnée relié au port B

Les deux registres reliés au port A et B sont des registres de donnée que l'on peut programmer ; ils ont quelques tâches dans l'ATARI ST, qui n'ont en réalité rien à voir avec la génération de sons. Ainsi le port B est directement relié au port parallèle de l'imprimante et le gère, alors que le port A s'occupe de la selection des unités de disquette, de la demande des données pour l'interface série et du port GPO pour le branchement du moniteur. La manipulation de ces registres n'est donc pas conseillée. Mais revenons à notre sujet : la génération de sons.

Si l'on a fait maintenant suffisamment d'essais pour obtenir les effets sonores voulus, on peut alors simplifier énormément leur utilisation. Il suffit pour cela de créer une table pour les registres et leurs contenus, de la stocker à une certaine place de la mémoire et de la viser avec le pointeur de la routine d'interruption-sound mentionné ci-dessus. De cette manière, toute une suite de sons pourra être générée grâce à une seule instruction, pendant que le programme BASIC continuera à travailler. On peut ainsi faire jouer une musique de fond, ou bien, comme dans les jeux, entendre le bruit d'un coup de feu ou d'une explosion, sans pour autant interrompre le programme.

Cependant, nous n'avons pas encore épuisé toutes les possibilités qu'offre cette technique de génération de sons et de bruits.

C'est ainsi qu'au cours d'un "morceau de musique", on peut aussi programmer des pauses qui, de la même façon, ne font intervenir que le générateur de sons et non pas notre programme BASIC. On peut également directement préprogrammer un niveau sonore ascendant ou descendant, pour réaliser, par exemple, le bruit d'une sirène à peu de frais. Considérons à ce sujet le programme suivant, qui produit un bruit caractéristique dont l'intensité sonore baisse :

```
10  rem ** mise en place d'une séquence de sons **
20  def seg = 0
30  defdbl b : b=2694           : rem déclaration d'un mot long
40  a$=space$(100)              : rem faire de la place
50  a=peek(varptr(a$)+2)        : rem déterminer l'adresse
60  def seg = 1
70  for i=0 to 100
80  read x                      : rem transmettre les valeurs
90  poke a+i-1,x
100 if x=255 then 120
110 next i
120 poke a+i,0                  : rem zéro final
130 def seg = 0
140 poke b,a                    : rem générer le son !

150 rem — données de la séquence sonore —
160 data 0,1,1,0,2,0,3,0
170 data 4,0,5,0,6,0,7,254
180 data 8,16,9,0,10,0,11,0
190 data 12,35,13,10

195 data 128,50
200 data 129,0,1,250

205 data 200,30

210 data 0,00,1,2,2,0,3,0
220 data 4,0,5,0,6,0,7,246
230 data 8,16,9,0,10,0,11,0
240 data 12,62,13,9
250 data 255
```

Le premier bloc de data compris entre les lignes 160 et 190 permet de produire un son dont l'intensité baisse, en le modulant par une courbe en forme de triangle. La dépendance du son avec la courbe enveloppe est obtenue en attribuant 16 (bit 4 à 1) au registre 8. La courbe enveloppe, quant à elle, est définie dans les registres 11, 12 et 13 du Sound-Chip.

Venons en aux lignes 195 et 200. Ici, on utilise deux instructions exceptionnelles en ce qui concerne les interruptions sonores. Une telle instruction exceptionnelle est indiquée à la place du numéro de registre, et a toujours une valeur supérieure à 127. L'instruction exceptionnelle 128 a pour effet de bufferiser la valeur suivante (rien de plus). Cependant, cette valeur n'a d'intérêt qu'en liaison avec l'instruction exceptionnelle suivante. Celle-ci, qui porte le numéro 129, exécute le processus suivant :

Le nombre qui suit est d'abord considéré comme un numéro de registre. Le registre est selectionné, puis on lui affecte la valeur qui a été mémorisée temporairement avec l'instruction précédente 128. Dans notre exemple, il s'agit de la durée de la période du canal A, donc de sa fréquence. On génère ainsi un son dont l'intensité varie suivant la courbe enveloppe.

Puis, on prend le nombre qui suit le numéro de registre et on l'ajoute de façon permanente à ce même registre. Le son devient donc de plus en plus grave, car une augmentation constante de la durée de la période correspond à une diminution constante de la fréquence.

Le dernier nombre, qui intervient dans l'instruction exceptionnelle 129, est la valeur limite que doit recevoir le registre. Dans notre exemple, il s'agit du son le plus grave que peut atteindre le canal A.

Si le Timing de l'exemple précédent est correctement réalisé, le fréquence du son est minimale au moment même où la courbe enveloppe atteint son maximum. Si l'on imagine un corps en train de tomber, celui-ci devrait maintenant rebondir. Pour maintenir un certain 'suspens', on introduit d'abord une petite pause avant que le choc n'arrive. Cette pause est réalisée par la troisième instruction exceptionnelle. Cette dernière a un numéro quelconque supérieur à 129. Le nombre qui suit indique la durée de la pause.

Pendant ce temps, l'état des registres du Sound-Chip et donc le son lui-même restent inchangés. On peut ainsi, de la même manière qu'avec l'instruction WAVE, mettre en place un retard avant qu'un son ne retentisse.

Avec cette astuce, la génération de sons grâce aux interruptions, une multitude de possibilités sont ouvertes pour réaliser une atmosphère 'bruyante' dans un jeu. On doit cependant remarquer qu'une touche enfoncée change la valeur du pointeur et interrompt la séquence programmée. Cela est également vrai pour les bruitages réalisés avec SOUND et WAVE. Pour terminer, nous allons encore examiner ces deux instructions.

L'instruction SOUND peut recevoir 4 ou 5 paramètres. Ces valeurs ont, pour l'instruction SOUND A,B,C,D,E, la signification suivante:

A	Numéro de canal	(0-2)
B	Intensité	(0-15)
C	Note	(1-12)
D	Octave	(1-8)
E	Durée	(0-255 en 1/50 de seconde) facultatif

La fréquence est donc divisée en octaves et en notes, de telle sorte que la transcription d'un morceau de musique se fait sans difficultés.

L'instruction WAVE nécessite également 4 à 5 paramètres. WAVE A,B,C,D,E signifie :

A Configuration. On utilise ici le registre 7 du Sound-Chip. En réalité, seuls les 6 bits de poids faible sont pris en compte.
B Courbe enveloppe (on/off)
C Forme de la courbe enveloppe, registre 13
D Durée de la période la la courbe enveloppe
E Retard avant le son (facultatif)

Avec ces deux instructions BASIC, vous pouvez à présent essayer de réaliser le bruit de chute de l'exemple précédent. Le nombre d'effets sonores que l'on peut obtenir avec l'ATARI ST est étonnant !

7.3 PROGRAMMATION DES FENETRES ET DES MENUS

Ce qui distingue l'ATARI ST des autres ordinateurs est très certainement la facilité d'utilisation qu'offre le travail avec les menus. En outre, on peut soi-même configurer son propre bureau. Les programmeurs en BASIC peuvent également tirer profit de cette technique.

La méthode la plus simple pour réaliser un menu est de faire correspondre des options de menus à des numéros, et de les afficher à l'écran. L'utilisateur est alors invité à introduire son choix. Le programme peut ensuite exécuter la partie de programme qui correspond à l'option choisie.

Un tel programme ressemble à :

```
10  print '1) Saisie des données'
20  print '2) Impression des données'
30  print '3) Fin'
40  input 'Votre choix ';c
60  on c goto 100,200,80
70  goto 40
80  end
100 rem ** saisie des données **
110 .....
200 rem ** impression des données **
210 .....
```

Cette méthode de gestion des menus est utilisée dans de nombreux programmes. Cependant, l'ATARI ST offre la possibilité de le réaliser de manière bien plus confortable.

Il y a tout d'abord les 10 touches-fonctions. Avec elles, on peut effectuer un choix similaire à l'exemple précédent, mais qui est plus confortable dans son utilisation. En effet, il n'est plus nécessaire de valider le numéro choisi par un Return, si l'on remplace l'instruction INPUT par :

```
50 c=inp(2)-186
```

Comme la touche F1 a pour valeur ASCII 187, la soustraction ci-dessus donne 1, ce qui permet de poursuivre avec l'instruction conditionnelle de branchement ON...GOTO. Ceci constitue donc la première étape de la simplification quant à la réalisation des menus. Cette méthode est également déjà utilisée dans beaucoup de programmes, comme par exemple le programme de traitement de texte SM-TEXT, avec lequel ce livre été écrit.

Mais faisons un pas supplémentaire qui nous conduira vers le menu 'parfait'. Le système d'exploitation de l'ATARI ST offre avec le GEM quelques possibilités concernant la gestion des menus. Pour la programmation du GEM, nous avons déjà utilisé l'instruction VDISYS. Mais le VDI ne nous est ici pas d'un grand secours, car il s'occupe uniquement des fonctions graphiques. Nous allons maintenant examiner à la loupe l'instruction encore mystérieuse GEMSYS.

GEMSYS active, de la même manière que VDISYS, le GEM de l'ATARI ST. La différence réside dans le fait que VDISYS n'appelle que le VDI alors que GEMSYS s'applique à l'AES. L'AES s'occupe de la gestion des fenêtres et des menus dans le ST. C'est lui qui permet à l'utilisateur de se déplacer en 'première classe' grâce à l'utilisation de la souris.

De la même manière que l'instruction VDISYS, GEMSYS nécessite l'emploi de certains paramètres lui indiquant quoi faire. Ces paramètres sont également transmis dans des vecteurs, qui portent les noms de CONTRL, INTIN, INTOUT, ADDRIN et ADDOUT. "Super !, pense le programmeur VDI expérimenté, je connais déjà les trois premiers". Erreur ! Il y a en effet ici un problème : CONTRL, INTIN et INTOUT ne sont pas identiques aux vecteurs du VDI bien connus ! Ces champs de données se trouvent d'ailleurs à une autre adresse dans la mémoire.

La question est donc : comment accède-t-on à ces adresses ?

Les variables systèmes du BASIC pour CONTRL et les autres ne peuvent pas être utilisées ; de plus le BASIC ne connait pas ADDIN et ADDOUT. Ce qu'il connait quand-même encore et qui a échappé à notre attention est la variable système GB. Il s'agit également d'un pointeur. Il ne pointe cependant pas sur un champ de données, mais sur une table qui contient à nouveau des pointeurs. Ce sont précisément de ces pointeurs que nous avons besoin.

Il s'agit de pointeurs de 32 bits, dont la signification est la suivante :

PEEK(GB)	CONTRL
PEEK(GB+4)	GLOBAL
PEEK(GB+8)	INTIN
PEEK(GB+12)	INTOUT
PEEK(GB+16)	ADDRIN
PEEK(GB+20)	ADDROUT

Ces pointeurs sont ensuite utilisés pour le transfert des paramètres à l'AES ou pour l'instruction GEMSYS. Le vecteur GLOBAL ne joue ici qu'un rôle secondaire. Il contient des informations sur le statut momentané de la commande GEM-AES en train d'être exécutée.

Voici la signification des différents éléments du vecteur GLOBAL :

GLOBAL	Numéro de la version du GEM
GLOBAL+2	Nombre maximal de programmes actifs simultanément
GLOBAL+4	Numéro du programme actuel
GLOBAL+10	Pointeur sur une structure d'arbre

Nous pouvons généralement oublier ce vecteur. Plus importants sont les autres tableaux. Parmi ceux-ci, le champ CONTRL contient les mêmes éléments que celui du VDI. CONTRL(0) contient à nouveau le numéro de fonction, CONTRL(1) et CONTRL(2) le nombre des éléments de INTIN et de INTOUT, et CONTRL(3) et CONTRL(4) le nombre des éléments de ADDRIN et de ADDROUT.

Appliquons donc l'instruction GEMSYS, en commençant par un appel très simple. Il s'agit, dans l'exemple suivant, de la fonction FORM_ERROR, qui affiche un message d'erreur du TOS dans une petite fenêtre au milieu de l'écran. Le numéro de l'erreur est affecté à INTIN(0).

```
10 rem ** Démo - TOS-ERROR **
20 defdbl b : b=gb
30 cn= peek(b)          : rem pointeur du vecteur CONTRL
40 ii= peek(b+8)        : rem pointeur du vecteur INTIN
50 poke cn,53           : rem numéro de fonction
60 poke cn+2,1
```

```
70  poke cn+4,1
80  poke cn+6,0
90  poke cn+8,0
100 poke ii,22          : rem numéro de l'erreur
120 gemsys 53           : rem et affichage
```

Regardons cet exemple plus en détail. On remarque tout d'abord qu'on ne travaille pas directement avec GB, mais par le biais d'une variable B. La raison à cela est la suivante : les pointeurs CN et II doivent être des mots longs. Un PEEK(GB) ne retournerait cependant que le mot HI du pointeur CONTRL, avec lequel on ne peut rien entreprendre. Mais comme la variable B est déclarée comme un mot double, PEEK(B) renvoie un mot long.

On pourrait ensuite se demander : pourquoi CONTRL et INTIN ne sont pas utilisés comme des noms de variables, ce qui clarifierait le programme ? Malheureusement, ces deux noms sont déjà reservés pour des variables systèmes, elles ne sont donc pas utilisables pour des applications d'un type différent.

Une autre différence avec la programmation du VDI apparait ici. L'instruction GEMSYS est directement écrite avec le numéro de fonction. La valeur suivant GEMSYS n'est donc pas ici un 'Dummy', mais bien le numéro de fonction lui-même.

L'affichage d'un message TOS-ERROR n'est hélas pas particulièrement utile dans un programme BASIC. Il serait plus intéressant d'afficher un autre texte et peut-être de mettre à disposition un plus grand nombre d'options de menu. Ceci est possible grâce à la fonction-AES FORM_ALERT, qui est normalement uniquement prévue pour les messages d'erreurs, mais qui est cependant utilisable pour d'autres applications. Deux paramètres sont transmis à cette fonction. Le premier est un pointeur sur un texte qui contient la description, les énoncés (au maximum 3) des options de menu et le numéro des symboles à représenter à coté des messages.

Ce texte peut facilement être affecté à une variable chaîne de caractères. On communique ensuite à la fonction AES l'adresse de cette variable. Par ailleurs, il faut encore transmettre un autre paramètre, qui détermine laquelle des options peut être choisie avec la touche 'Return'. Celle-ci est alors fortement encadrée sur l'écran, comme dans l'exemple connu du choix entre LOAD ou SAVE.

Pour cela, on utilise l'option OK qui signifie qu'une action sur la touche Return doit provoquer l'exécution de l'opération .

Il y a ici cependant un problème. La fonction retourne à INTOUT(0) le numéro de l'option choisie. Avant de pouvoir lire cette valeur par PEEK(IO), elle est modifiée par un autre appel qui est effectué par le BASIC. Je n'ai ainsi pas réussi à obtenir ce numéro. La seule possibilité qui nous reste est d'affecter 0 à INTIN(0). Cela implique qu'aucune des options de choix n'est validable par la touche Return. De ce fait, ce choix ne peut s'effectuer que par l'intermédiaire du curseur de la souris, dont la position peut être déterminée et évaluée immédiatement après le retour de la fonction AES. On peut laisser de coté la coordonnée suivant Y du curseur de la souris, car les options sont délibérément affichées les unes à côté des autres.

L'appel de la fonction permettant de créer le texte à afficher est réalisé dans le programme suivant :

```
10  rem  ** Demo - alert **
11  defdbl b,d : b=gb
12  cn=peek(b)          : rem définir le pointeur
14  ii=peek(b+8)
20  io=peek(b+12)
25  d=peek(b+16)
30  a$ = '<1><Vous avez le!choix : .....>'
31  a$=a$+'<Touche 1!Touche 2!Touche 3>'
40  a=peek(varptr(a$)+2)
50  poke cn,52          : rem numéro de fonction
60  poke cn+2,1
70  poke cn+4,1
80  poke cn+6,1
90  poke cn+8,0
100 poke ii,0           : rem pas de choix avec 'return'
110 poke d,a            : rem adresse du texte
120 gemsys 52           : rem et exécution
125 t=peek(io)          : rem normalement, numéro de touche...
```

Les crochets qui entourent le texte devraient en fait être des crochets droits. De la même façon, les points d'exclamation jouent le rôle du trait vertical, qui est obtenu en appuyant à la fois sur la touche shift et sur la touche en dessous de DELETE. Veuillez remplacer ces caractères par les caractères correspondants.

Le 1 à l'intérieur des premiers crochets indique uniquement le symbole qui doit se trouver à côté du texte descriptif. Vous avez le choix entre des symboles comme un panneau STOP, un point d'interrogation ou un point d'exclamation.

Le crochet suivant contient le texte descriptif. Le passage à la ligne suivante est obtenu par le trait vertical. La longueur du texte devrait être choisie de telle manière que la fenêtre contenant les options de choix soit bien proportionnée. Ces options sont définies grâce au contenu du troisième crochet. Il doit y avoir au maximum 3 choix possibles, chaque inscription ne devant pas dépasser 20 caractères. La séparation des différentes inscriptions est à nouveau indiquée par les traits verticaux.

La ligne 125 du programme ci-dessus peut être supprimée. Elle n'aurait un sens que si le BASIC ne nous modifiait pas la valeur de INTIN(0). L'évaluation de la touche enfoncée doit donc malheureusement être effectuée à la suite.

Pour interroger la position de la souris, nous connaissons déjà plusieurs méthodes. Mais comme nous sommes précisément en train de parler du GEM-AES, nous allons utiliser une instruction GEMSYS pour exécuter cette scrutation.

La fonction correspondante ne retourne pas uniquement au programme appelant la position du curseur de la souris. On obtient en outre l'état des deux touches de la souris ainsi que des touches Shift, Control et Alternate. L'appel ne nécessite aucun transfert de paramètres.

```
10   rem ** état de la souris **
11   defdbl b : b=gb
12   cn=peek(b)
25   io=peek(b+12)
50   poke cn,79          : rem numéro de la fonction
60   poke cn+2,0
```

```
70  poke cn+4,5
80  poke cn+6,0
90  poke cn+8,0
100 gemsys 79          : rem appel de l'AES
110 for i=2 to 8 step 2
120 x(i)=peek(io+i) : next i
130 print ' Position       ';x(2);' , ';x(4)
140 print ' touches-souris ';x(6)
150 print ' touches        ';x(8)
```

Le détour par la variable X est nécessaire, car un PRINT, qui par exemple afficherait directement la position de la souris, appelerait lui-même l'AES et modifierait de ce fait les paramètres. Il est intéressant de noter qu'après l'appel de la fonction 52, le vecteur INTOUT est détruit par la fonction 79 elle-même. De ce fait, le calcul de la position de la souris peut suivre sans instruction GEMSYS supplémentaire.

Les fonctions AES présentées ci-dessus créent automatiquement une petite fenêtre sur l'écran, sans que l'on ait à se casser la tête sur la taille ou la position de la fenêtre. Le BASIC propose cependant quatre fenêtres différentes, dont la taille et la position sont modifiables à l'aide du curseur de la souris. Pour modifier une de ces fenêtre dans un programme, il faut à nouveau faire appel à l'AES. Il s'agit de la fonction 105 qui porte le nom de WIND_SET. Grâce à elle, on peut non seulement déplacer une fenêtre, mais également définir les caractéristiques de la fenêtre. Ainsi, on peut par exemple empêcher qu'elle soit effacée par la souris.

Voici un exemple de programme, qui met en place la taille et la position de la fenêtre LIST. En affectant une valeur différente à la ligne 40, on peut aussi modifier une des autres fenêtres.

```
10 rem ** mise en place d'une fenêtre **
20 defdbl b : b=gb
30 ii=peek(b+8)
40 h=2                 : rem fenêtre LIST
50 poke ii,h           : rem choix
60 poke ii+2,5         : rem mode
70 poke ii+4,20        : rem abscisse X
80 poke ii+6,20        : rem ordonnée Y
```

```
90  poke ii+8,350       : rem largeur de la fenêtre
100 poke ii+10,100      : rem hauteur
110 gemsys 105          : rem et mise en place
```

Les coordonnées X et Y indiquent la position du coin supérieur gauche de la fenêtre. Le mode 5, qui est affecté ligne 60, indique la fonction que doit exécuter l'AES.

Les différents modes possibles sont les suivants :

Mode 1 : En ii+4 est affecté un nombre dont les différents bits définissent les caractéristiques de la fenêtre.

Voici la signification de chaque bit :

bit 0 : ligne du titre de la fenêtre
bit 1 : effacement
bit 2 : ouverture
bit 3 : déplacement
bit 4 : ligne d'information
bit 5 : modification de la taille
bit 6 : flèche vers le haut
bit 7 : flèche vers le bas
bit 8 : curseur vertical
bit 9 : flèche vers la gauche
bit 10 : flèche vers la droite
bit 11 : curseur horizontal

Mode 2 : Le nom de la fenêtre est modifié. L'adresse du nouveau nom est affecté dans un mot long à l'adresse ii+4 jusqu'à l'adresse ii+7. Le texte doit se terminer par un zéro.

Mode 3 : La ligne d'information de la fenêtre est modifiée. Les conditions sont les mêmes que pour le mode 2.

Mode 5 : La fenêtre est mise en place. Ce mode est utilisé dans notre exemple.

Mode 8 et 9 : La position relative du curseur horizontal ou du curseur vertical est mise en place.

Mode 10 : La fenêtre actuelle est choisie

Mode 15 et 16 : La taille relative du curseur horizontal ou du curseur vertical est mise en place.

Grâce à ces fonctions, on peut donc donner à n'importe quelle fenêtre la forme souhaitée. Un programme BASIC qui gère des fenêtres est ainsi plus souple à réaliser.

7.4 TRAITEMENT DE TEXTE

Les textes jouent un très grand rôle dans les applications d'un ordinateur. Un ordinateur n'est en effet pas uniquement destiné à effectuer des calculs. Il peut également gérer, modifier, mémoriser et imprimer des textes. Mais les textes, dans notre cas les variables chaînes de caractères, peuvent aussi servir à d'autres emplois. On peut y affecter des données d'un type différent comme par exemple un programme machine. Nous avons déjà utilisé ce principe.

Pour affecter une zone mémoire à une chaîne de caractères, on peut utiliser le pointeur de variable VARPTR() et POKE. VARPTR(A$) par exemple pointe sur un tableau, où se trouvent quelques informations concernant la variable A$. Le deuxième élément de ce vecteur représente l'adresse mémoire de la variable elle-même. On obtient donc cette adresse par l'instruction A=PEEK(VARPTR(A$)+2). Pour charger dans cette variable, aussi appelée String, un fichier contenu sur disquette, on peut utiliser le programme suivant :

```
10 rem ** affectation d'un fichier à une variable string **
15 a$=space$(200)                    : rem réserver de la place
17 a=peek(varptr(a$)+2)              : rem déterminer l'adresse
20 input 'Nom du fichier ';nf$       : rem saisir le nom du fichier
30 bload nf$,a                       : rem charger
40 print :o=(peek(a) and 255)        : rem calculer l'offset
50 f=o+peek(a+4)                     : rem calculer la fin
60 for i=o+2 to f step 2
70 print i, peek(a+i)                : rem afficher le contenu
80 next i
```

Avec ce programme, on peut affecter à A$ des petits fichiers contenus sur disquette (< 200 octets). Les lignes 40 à 80 ne sont pas indispensables , elles n'ont de sens qu'en liaison avec un programme d'application (*.PRG). Si on charge un tel programme, on calcule alors ses adresses de début et de fin, et on l'affiche sous forme de mots décimaux. J'ai utilisé ce programme pour transposer les petits programmes créés en assembleur dans des lignes de data. A cet effet, il suffit d'entrer les nombres, qui ont été affichés, dans des lignes de data, afin de pouvoir ensuite à nouveau les affecter à la chaîne de caractères avec une boucle READ-POKE.

On peut également appliquer cette méthode de stockage dans une variable pour sauvegarder un plus grand texte sur disquette. On peut certes aussi le faire avec OPEN et PRINT#, mais cela dure bien plus longtemps qu'avec BLOAD et BSAVE.

Cependant, il ne faut pas oublier les véritable utilisation des variables chaînes de caratères. L'une de ces utilisations est de mémoriser des enregistrements et éventuellement de pouvoir les trier. Un tel programme de tri est vraiment facile à programmer avec le BASIC de l'ATARI ST. Ce dernier connait en effet des instructions telles que SWAP.

Voici un exemple dans lequel l'algorithme appelé Bubble-Sort est utilisé :

```
10  rem ** Tri de textes **
20  dim m$(10)
30  for x=1 to 10
40  print 'Mot Nr ';x;' :';:input m$ (x)
50  next x
60  print 'Le processus de tri est en cours'
70  for i=1 to 10
80  for j=1 to 10
90  if m$(j)<m$(i) then swap m$(i),m$(j)
100 next j
110 next i
120 for x=1 to 10:print m$(x): next x
```

On entre tout d'abord 10 mots, qui sont ensuite affectés au vecteur de chaînes de caractères M$().

Puis le tri s'effectue, en comparant les mots dans une boucle et en intervertissant éventuellement leurs places grâce à l'instruction SWAP.

Les chaînes de caractères dans le programme ci-dessus peuvent être par exemple des noms, des adresses ou aussi des numéros de téléphone, qui sont ordonnés dans un fichier. On peut aussi appliquer cette routine de tri pour des textes plus volumineux, comme par exemple des tableaux de string de plusieurs dimensions. Un exemple d'une telle application est facile à trouver : le Desktop peut en effet trier les enregistrements du répertoire de la disquette suivant différents critères, comme par exemple le nom, le type, ou la taille. On pourrait réaliser de la même manière un programme BASIC de gestion des coordonnées individuelles, dans lequel les enregistrements seraient triés suivant le nom, l'adresse ou le numéro de téléphone. La saisie de ces données est plus agréable si l'on utilise un masque de saisie.

7.4.1 Masque de saisie

Les masques de saisie sont utilisés dans tous les programmes usuels de gestions de données. On affiche un 'écran' prédéfini, dans lequel on réserve de la place pour les enregistrements à saisir. L'utilisateur ne peut alors entrer des données que dans les emplacements prévus. L'écran terminé ressemble souvent exactement au formulaire que l'on peut ensuite imprimer.

```
10  rem ** masque de saisie **
20  dim t$(100,6),x(7),y(7)       : rem définir les tableaux
30  fullw 2 : clearw  2           : rem préparer la fenêtre
40  for i=1 to 7
50  read x(i),y(i),a$             : rem lire le masque
60  gotoxy x(i),y(i)              : rem placer le curseur
70  print a$;left$('.....',11-len(a$));' :'
80  next i
90  data 10,1,'** Saisie de l'adresse **'
100 data 1,5,'Nom'
110 data 16,5,'Prénom'
120 data 1,10,'Ville'
130 data 1,12,'Rue'
140 data 1,15,'Telephone'
```

```
150 data 2,18,'Remarque'
160 n=1                              : rem première saisie
165 gotoxy 1,1 : print n;' )'        : rem écrire le numéro
170 for i=1 to 6
180 gotoxy x(i+1)+6,y(i+1)           : rem placer le curseur
190 input t$(n,i)                    : rem saisie
195 if t$(n,i)='#' then i=10         : rem fin !
200 next i
210 if i<10 then n=n+1 : goto 165
220 n=n-1                            : rem corriger le nombre n
```

Dans ce programme, on crée un masque de saisie qui est ensuite affiché à l'écran ou dans la fenêtre OUTPUT. Puis on effectue la saisie proprement dite. Si on entre dans n'importe quel enregistrement le caractère #, la saisie est interrompue. Dans les lignes suivantes, on mémorise les données enregistrées, en affectant le n-ième nom à T$(n,1), le n-ième prénom à T$(n,2), et ainsi de suite. La variable N contient le nombre de saisies effectuées.

On peut maintenant charger la routine de tri précédente. Si, par exemple, on veut trier suivant le numéro de téléphone, on utilisera uniquement T$(n,5).

7.5 PILOTAGE : SOURIS/JOYSTICK

Pour influencer le déroulement d'un programme, on utilise en général le clavier. Cependant, pour certaines applications, il peut être gênant de se servir du clavier pour valider uniquement une opération. C'est par exemple le cas dans certains jeux : il serait commode de pouvoir répondre avec le joystick à la question demandant si l'on veut jouer une nouvelle partie ou non.

Un tel pilotage n'est pas difficile à programmer. Une question Oui/Non peut être validée par un mouvement gauche/droite de la manette du joystick ou par une action sur la touche gauche ou droite de la souris. Cela se complique quand il y a plusieurs choix : on peut par exemple afficher un menu dont l'utilisation est possible avec la souris ou le Joystick.

Voici un programme utilisant cette méthode :

```
10  rem ** choix avec joystick 1 **
15  def seg = 1                 : rem PEEK et POKE adressent un octet
20  clearw 2 : gotoxy 0,1       : rem préparer la fenêtre
30  print ' 1.':print' 2.':print' 3.'  : rem présenter le menu
40  y=peek(3592)                       : rem joystick vertical
45  poke 3592,0                        : rem remettre à zéro
50  w=w-(y=1)+(y>1)-(w>3)+(w<1)         : rem nouvelle position
60  gotoxy 0,w : print '=>'             : rem indiquer le choix
70  for i=1 to 100 : next i            : rem pause
80  goto 0,w : print ' '                : rem effacer l'index
90  if (peek(3581)and 3)=0 then 40
```

Ici, on déplace de haut en bas un index (flèche) entre trois options. On quitte le menu en actionnant le bouton de feu. La variable W contient alors le numéro du point de menu choisi. On peut ainsi traiter un nombre quelconque d'options de menu, aussi longtemps que celles-ci tiennent sur l'écran.

Voici maintenant la version pour la souris :

```
10  rem ** choix avec la souris **
20  clearw 2 : gotoxy 0,1              : rem préparer la fenêtre
30  print ' 1.':print ' 2.':print' 3.' : rem afficher le menu
40  y=peek(8996)                       : rem position de la souris
50  w=1-(y>70)-(y>90)                  : rem déterminer le choix
60  gotoxy 0,w : print '=>'            : rem et indiquer
70  for i=1 to 100 : next i            : rem pause
80  gotoxy 0,w : print ' '             : effacer l'index
90  if (peek(2633)and 3)=0 then 40
```

Ce programme calcule la position choisie pour la flèche à partir de la position verticale du curseur de la souris. Ici, on quitte le menu en appuyant sur l'une des deux touches de la souris. La représentation de l'index est en réalité inutile ; l'index de la souris suffirait. Une telle flèche est cependant plus sûre pour ne pas obtenir de faux résultats de calcul à cause d'une petite imprécision de positionnement.

La position du curseur de la souris peut naturellement aussi être évaluée en deux dimensions. La position horizontale de la souris est normalement mémorisée à l'adresse 8994, la position verticale à l'adresse 8996. Ces deux mots mémoires peuvent être manipulés avec POKE, afin de placer le curseur à une position quelconque de l'écran. Pour amener le curseur de la souris dans le coin supérieur gauche de l'écran, il suffit de taper ou de programmer POKE 8994,0 et POKE 8996,0. Cela est intéressant quand, par exemple, on veut réaliser un dessin à l'écran et que l'on veut commencer son tracé à une position définie.

7.6 ENTREES/SORTIES

Après avoir utiliser de toutes les manières possibles la mémoire et l'écran de l'ordinateur, on peut maintanant s'intéresser à la communication avec des appareils externes. Il s'agit en particulier des unités de disquettes et de l'imprimante.

Les entrées/sorties classiques sur ces appareils ont déjà été étudiées. Cependant, les possibilités de ces périphériques n'ont pas encore été épuisées. Les disquettes sont divisées en secteurs auquels on peut accéder séparément de manière très simple. De même, en ce qui concerne l'imprimante, il existe certaines fonctions que nous allons également étudier. Enfin, nous allons examiner l'emploi et la programmation d'un modem téléphonique.

Les différentes interfaces et ports de sortie de ces périphériques sont déjà connus. Nous pouvons donc nous occuper plus en détail de la programmation des appareils une fois connectés.

7.6.1 Pilotage de l'imprimante

Pour pouvoir imprimer quelque chose, on peut utiliser les instructions bien connues LPRINT et LLIST. En outre, nous avons la possibilité d'envoyer des données au moyen de l'instruction OUT 0,x par laquelle un caractère quelconque peut être transmis à l'imprimante. Cela ne se limite pas à l'envoi de caractères alphanumériques ; des caractères de contrôle peuvent également être transmis à l'imprimante. L'instruction OUT 0,x correspond en fait exactement à l'instruction LPRINT CHR$(x); , mais elle est plus simple à écrire et par ailleurs plus rapide.

Si l'on appuie maintenant sur la touche Alternate et la touche Help, l'ordinateur envoie, lors de l'impression hardcopy, quelques caractères de contrôle. Ceux-ci indiquent d'une part le mode d'impression afin que l'imprimante n'interprète pas les valeurs transmises comme des valeurs ASCII et imprime n'importe quoi. D'autre part, on indique à l'imprimante de mettre en place la gestion des lignes pour que chaque ligne graphique ait un certain écart avec la ligne précédente. Un des problèmes pouvant ici apparaître est la non compatibilité d'une imprimante. Cela veut dire que certaines imprimantes possèdent d'autres caractères de contrôle pour ces mêmes fonctions. La reconfiguration de l'interface imprimante à partir du Desktop n'est alors d'aucune aide.

Le système d'exploitation, qui est chargé à partir de la disquette, prend les caractères de contrôle nécessaires dans une table qui se trouve en mémoire à partir de l'adresse $16D5E. Cette table est naturellement modifiable et on peut ainsi adapter une autre imprimante au ST.

Les différents éléments de cette table sont les suivants :

$16D5E	Esc 'X' 6	
$16D63	Esc 'X' 5	
$16D68	Esc 'X' 3	Imprimante couleur matricielle pour ATARI
$16D6D	Esc 'L'	Imprimante matricielle S/W : 960 points par ligne.
$16D71	Esc 'Y'	Imprimante couleur : 960 points par ligne
$16D75	Esc '3' 1	Interligne de 1/216 pouce
$16D7A	Esc '3' 1	Idem
$16D7F	Esc '1'	Interligne de 7/72 pouce
$16D83	Esc '2'	Interligne de 1/6 pouce.

Escape L indique à l'imprimante, la EPSON-S/W par exemple, qu'elle va recevoir des masques de bits qui doivent directement être imprimés. Après ce caractère, il y a la valeur indiquant le nombre d'octets à envoyer pour la ligne. Puis, le masque de bits de chaque octet suivant est imprimé l'un à côté de l'autre dans la colonne courante. Si votre imprimante a, pour cette application ou pour une autre, un autre caractère de contrôle, vous pouvez le modifier en le POKant à l'adresse correspondante.

L'impression de l'écran peut être obtenue en appuyant simultané sur les touches correspondantes, mais aussi en choisissant l'option de menu 'Impression de l'écran' ou en entrant POKE 1262,0. Le système d'exploitation gère ensuite complêtement l'impression, aussi longtemps que les touches Alternate et Help ne sont pas à nouveau enfoncées.

Pour entreprendre la programmation de l'imprimante elle-même, par exemple pour imprimer des caractères que vous avez vous-même définis, il faut transmettre la séquence d'instructions adéquate à l'imprimante. En outre, on a le choix de la résolution avec laquelle l'imprimante doit travailler. Escape L permet d'imprimer en double densité ; cela est nécessaire pour l'impression de cet écran de très haute résolution. De même on peut, avec une imprimante EPSON-S/W ou une imprimante compatible, choisir la résolution normale, en utilisant Escape K.

Voici un petit programme qui permet d'imprimer un caractère redéfini :

```
10 rem ** création et impression d'un caractère **
20 for i=1 to 16
30 read a                    : rem lire le caractère
40 out 0,a                   : rem et l'envoyer
50 next i
60 lprint                    : rem fin, saut de ligne
70 data 27,75,12,128
80 data 4,10,26,58,103,231
90 data 231,103,58,26,10,4
```

Cet exemple donne à l'imprime un petit OVNI. Le 12 indique le nombre de masques à imprimer. Si ce nombre est supérieur à 255, il faut ajouter l'octet de poids fort à 128. Puis viennent les données elles-mêmes, dont les bits de poids faible sont imprimés en bas.

7.6.2 Utilisation des disquettes

Pour utiliser les disquettes comme intermédiaire de stockage pour nos données de programme, nous pouvons employer les instructions LOAD et SAVE ou BLOAD et BSAVE, comme cela est décrit au paragraphe 2.2. Tout le pilotage du lecteur de disquette est ensuite effectué par le BASIC ou le GEMDOS.

Il est cependant possible de passer en 'pilotage manuel' en court-circuitant le système d'exploitation. Pour cela, nous devons faire exécuter un programme machine qui peut à nouveau être placé dans une variable chaîne de caractères.

Dans ce paragraphe, nous allons examiner un tel programme, qui a en outre d'autres fonctions. La première partie de ce programme lit le programme machine dans des lignes de data et le place dans une chaîne de caractères. Nous avons déjà utiliser plusieurs fois cette méthode : aucune explication supplémentaire n'est donc nécessaire.

La deuxième partie du programme pose la question : quel secteur de la disquette doit être lu ? La valeur entrée au clavier est alors insérée dans le programme machine pour lui indiquer de quel secteur il s'agit. Les chaînes de caractères D$(0) et D$(1) sont déclarées pour recevoir les données du secteur. Ici, il est nécessaire d'utiliser deux variables l'une à la suite de l'autre, car une seule chaîne de caractères ne peut contenir plus de 255 octets. Ensuite, on affecte à la variable D l'adresse de début du buffer.

Enfin, on appelle le programme machine auquel il est nécessaire de transmettre en paramètre l'adresse du buffer. L'unité de disquette démarre et le contenu du secteur choisi est chargé dans le buffer. Comme chaque secteur contient toujours 512 octets, une partie des informations lues va au delà des variables D$(0) et D$(1). On peut naturellement utiliser un autre buffer de données. Dans ce cas, il ne faut pas oublier de se demander si cette zone mémoire n'est pas utilisée par un autre programme.

Quand le secteur est chargé en mémoire, on affiche les codes héxadécimaux et les caractères ascii correspondants. Cela devient intéressant si l'on veut par exemple explorer la structure de la directory, c'est-à-dire du répertoire de la disquette. Ce répertoire commence en face 0, sur la piste 1, et secteur 3.

```
10  rem ** accès disquette **
20  fullw 2                          : rem grande fenêtre Output
30  def seg=0                        : rem accès à des mots
40  a$=space$(42)                    : rem préparer le string
50  b=peek(varptr(a$)+2)             : rem déterminer l'adresse
60  for i=0 to 40 step 2
70  read a                           : rem lire les données
```

```
80  poke b+i,a                          : rem et charger en mémoire
90  next i
95  rem — données du programme machine —
100 data 8815,14,16188,1,16188,0
110 data 16188,1,16188,6,16188,0,17063,12041
120 data 16188,8,20046,-8196,0,20,20085
130 print : input "Face, Piste, Secteur";x,y,z
140 poke b+10,x                         : rem numéro de secteur
142 poke b+14,y
144 poke b+18,z
150 dim d$(4)                           : rem préparer le buffer
160 d$(0)=space$(255)
170 d$(1)=space$(255)
180 d=peek(varptr(d$(0))+2)             : rem déterminer l'adresse
190 call b (d)                          : rem charger le secteur !
200 def seg = 1                         : rem accès à des octets
210 print : for i=1 to 512
220 x% = peek(i+d-2)                    : rem lire l'octet
230 if x%<16 then print '0';            : rem uniquement pour le
                                          format
240 print hex$(x%);' ';                 : rem écrire l'octet en
                                          hexa
245 if (i and 15)>0 then 300            : rem ligne terminée ?
250 print ' '; : for j=17 to 2 step -1
260 x%=peek(i+d-j)
270 if x%<11 then ?'.'; : goto 290
280 print chr$(x%);                     : rem caractère Ascii
290 next j : print                      : rem caractère suivant
300 next i                              : rem ligne suivante
```

Le programme machine est très simple :

```
MOVE.L 14(SP),A1        * sauver l'adresse du buffer
MOVE.W #1,-(SP)         * nombre de secteur = 1
MOVE.W #0,-(SP)         * face 0
MOVE.W #1,-(SP)         * piste 1
MOVE.W #6,-(SP)         * secteur 6 de la piste
MOVE.W #0,-(SP)         * lecteur A
CLR.L    -(SP)          * ? (doit être)
MOVE.L A1,-(SP)         * adresse BUFFER
MOVE.W #8,-(SP)>        * Commando Floppy-Read
TRAP   #14              * appel XBIOS
```

```
ADDQ.l #20, SP        * restaurer la pile
RTS                   * retour au Basic
```

Ici, on définit non seulement le numéro de secteur, mais aussi le lecteur, le nombre de secteurs à charger et le sens de transmission des données. En ce qui concerne ce dernier paramètre, il est en effet possible d'écrire du buffer dans le secteur, en substituant la valeur 0 du sens des données par un 1. Attention cependant : de tels essais d'écriture peuvent avoir des effets désagréables sur la disquette, si le contenu du secteur est incorrect, et en particulier si l'on écrit par dessus le répertoire. Il est donc conseillé de travailler d'abord avec une disquette qui ne contient rien d'important.

Cette méthode permet donc de réaliser un accès direct à la disquette. Mais la méthode classique consiste à choisir un nom, sous lequel sont rangées les informations souhaitées sur la disquette. On entre ensuite un LOAD 'Chemin:Nom.Type', et le programme se charge. Le chemin désigne ici l'indication du lecteur à utiliser et/ou les noms des sous-directories du repertoire dans lequel se trouve le fichier. Le nom du fichier, quant à lui, ne doit pas dépasser 8 caractères. Le type qui se trouve en fin de descripteur est une abréviation constitué de trois caractères qui indique le type du fichier. Ce type peut être choisi indifféremment, mais certains types sont réservés. PRG signifie qu'il s'agit d'un programme machine exécutable, BAS est utilisé pour les programmes BASIC ,etc...

L'indication du type permet de selectionner et de rechercher plus facilement un programme ou un fichier. Si l'on cherche un programme BASIC, on obtient par un DIR *.BAS la liste des programmes disponibles.

C'est exactement de cette manière que l'on procède pour le choix des fichiers, lorsqu'on utilise les options 'chargement' ou 'sauvegarde' du menu. Il apparaît alors une fenêtre à l'écran qui est divisée en plusieurs fenêtres isolées. Examinons la ligne supérieure. C'est ici que se trouvent les critères de selection et le chemin qui déterminent les fichiers disponibles. Si par exemple cette ligne contient *.BAS, tous les programmes ayant comme suffixe type (Extender) .BAS sont recherchés.

L'étoile remplace ici tous les autres caractères qui peuvent se présenter. En modifiant cette ligne avec la souris, on peut par exemple afficher les programmes d'un autre lecteur de disquette.

La totalité de cette fenêtre et son utilisation sont entièrement gérées par l'AES du GEM. Nous pouvons donc sans difficultés appliquer cette méthode de selection de fichiers dans un programme BASIC. Cela peut s'avérer utile lorsqu'on a, par exemple, écrit un programme de gestion de fichiers et que l'on veut à présent charger l'un d'eux de la disquette. Pour pouvoir ensuite appeler la fonction qui affiche la fenêtre et qui réalise la selection des fichiers, nous devons préparer quelques paramètres.

Nous avons tout d'abord besoin de deux buffers. Dans le premier, on déposera le critère de selection mentionné ci-dessus ; le second contiendra le nom de fichier qui sera choisi dans la liste en appuyant sur la touche Break ou OK de la souris. En relation avec la définition du chemin contenu dans l'autre buffer, nous disposons alors du descripteur complet du fichier. Nous pouvons ensuite accèder à ce fichier pour la sauvegarde ou le chargement de nos informations.

Voici un exemple de programme qui appelle cette fonction AES :

```
10  rem  ** sélection de fichier **
20  poke contrl,122          : rem appel du VDI pour
30  poke intin,0
40  vdisys 0                 : rem mettre en place le curseur

50  defdbl b,d : b=gb
60  cn=peek(b)
70  d=peek(b+16)             : rem définition de l'adresse
80  io=peek(b+12)
90  a$='\*.PRG'              : rem définition du chemin
100 for i=1 to 40 : a$=a$+chr$(0)      : next
110 for i=1 to 15 : b$=b$+chr$(0)      : next
120 a=peek(varptr(a$)+2)
130 c=peek(varptr(b$)+2)
140 poke cn,90               : rem commande AES
150 poke cn+2,0
160 poke cn+4,2
170 poke cn+6,2
```

```
180 poke cn+8,0
190 poke d,a                    : rem charger le chemin
200 poke d+4,c                  : rem buffer contenant le nom
210 gemsys 90                   : rem appel

220 print 'Dans ';a$
230 print b$;' a été choisi'    : rem affichage du résultat
```

Le début de ce programme est un appel du VDI. Cette fonction remet en place le curseur de la souris, qui disparaît en effet quand une touche est enfoncée. Si l'on ne fait pas appel à cette fonction VDI et que l'on lance l'exécution du programme, le curseur reste invisible tout au long de l'exécution du programme. Cela constituerait un certain handicap pour le choix du programme...

A la suite de cela, on définit les pointeurs pour les paramètres AES. Vous trouverez l'explication de cette méthode dans le chapitre consacré à la programmation des menus et des fenêtres. Puis on initialise les deux buffers d'identification du fichier. Les deux boucles remplissent ces buffers, qui sont à nouveau des variables chaînes de caractères, avec la valeur 0. Cela est nécessaire pour un bon déroulement de la routine AES.

On transmet ensuite les paramètres dans la table des paramètres de l'AES. ADDRIN(0) contient en particulier les deux adresses des buffers qui sont des mots longs. Enfin, on procède à l'appel de l'AES du GEM lui-même. Après une pression sur l'une des deux touches de la souris ou un double-pression, le fichier pointé est validé, la fenêtre de choix est effacée et on rend la main au programme BASIC. Les deux variables chaînes de caractères contiennent à présent respectivement le chemin et le nom du fichier choisi.

La fonction transmet en outre l'indication de la touche de la souris (OK ou Break) qui a été actionnée pour interrompre cette même fonction. Cette information est envoyée à INTOUT(1). Malheureusement, le BASIC détruit cette valeur, avant que nous puissions la lire. Il faut donc pour cela calculer la position de la souris.

7.6.3 Communication

La transmission de données par voie téléphonique s'est developpée ces derniers temps de manière considérable pour devenir un véritable hobby. Il s'agit moins du dialogue entre deux ordinateurs que de la communication avec des centres serveurs. Nous allons tout d'abord examiner comment on peut entrer dans le 'monde de la télécommunication' avec l'ATARI ST.

En premier lieu, nous avons besoin d'un modem téléphonique. De tels appareils, qui réalisent la liaison entre le téléphone et l'ordinateur sont disponibles dans le commerce.

Un modem possède le plus souvent une sortie série permettant de le connecter à un ordinateur, et deux receptacles dans lesquels on place le combiné après obtention de la liaison. La transmission a ensuite lieu avec un sifflement aigu, car les signaux sont modulés en fréquence par l'électronique implantée dans le modem. La vitesse de transmission est de 300 baud, en ce qui concerne l'utilisation de 2 modems accoustiques reliés, car avec un débit plus élevé, la sécurité de de la transmission n'est plus assurée. Les modems, qui sont directement reliés au réseau commuté peuvent travailler avec des débits allant jusqu'à 1200 baud.

C'est pourquoi nous configurerons en premier lieu l'interface série sur 300 baud. Conmme la plupart des centres serveurs renvoient directement chaque caractère reçu en écho, nous choisiront également le mode de transmission Full Duplex.

Branchons à présent le modem à la sortie correspondante du ST. Pour pouvoir commencer la transmission des données, nous devons lancer un programme qui gère cette transmission. La méthode la plus simple consiste à appeler le Terminal-Emulator de l'ATARI ST. Chaque caractère que l'on entre au clavier est automatiquement envoyé à l'interface série ; de même, chaque caractère reçu est directement affiché à l'écran. Nous pouvons donc appeler tout de suite un centre serveur, poser l'écouteur sur le modem et dialoguer avec le centre serveur par l'intermédiaire du clavier et de l'écran.

Il y a cependant un hic : en effet, avec l'utilisation du Terminal-Emulator, nous n'avons pas la possibilité d'imprimer ou de mémoriser en mémoire notre 'conversation' avec le centre serveur.

Comme certains centres serveurs transmettent des textes d'introduction ou des menus d'aide qui sont souvent très longs, une partie des informations peut se perdre si l'on ne peut pas lire aussi vite que ne défilent les lignes à l'écran.

Il faut donc renoncer au confort de l'émulateur et écrire soi-même un petit programme qui gère la communication via modem, et mémorise en même temps les textes reçus. Un tel programme doit interroger en permanence l'interface et le clavier pour savoir si un caractère s'y trouve. De cette manière, on peut réaliser un environnement duplex, dans lequel l'envoi et la reception se déroulent de manière quasi simultanée.

Cependant, un autre problème se posa lors de la création du programme. La fonction INKEY$, avec laquelle on scrute habituellement le clavier sans pour autant interrompre le programme en cours, n'est pas utilisable avec la version BASIC dont on dispose actuellement. Si, à la place, nous prenons l'instruction INP(2), le programme attend jusqu'à ce qu'une touche soit enfoncée. Pendant ce temps, on peut certes recevoir des informations, mais on ne peut pas les lire !

Il faut donc recréer d'une manière ou d'une autre cette instruction. Pour cela, nous allons utiliser une astuce réalisable avec PEEK et POKE.

Voici comment :

```
10 rem ** Simulation de INKEY$ **
20 p=2552                    : rem pointeur de caractère
30 x=peek(p)                 : rem mémorisation intermédiaire
40 gotoxy 1,1 : print a      : rem par exemple
45 a=a+1
50 if peek(p)=x then 30      : rem nouveau caractère ?
60 x=peek(p)                 : rem si oui, nouveau pointeur
70 t=peek(2256+x) and 255    : rem chercher le caractère
80 print chr$(t)             : rem et afficher
90 goto 40                   : rem boucle sans fin
```

Chaque caractère saisi au clavier est automatiquement mis dans un buffer par le système d'exploitation. Ce buffer commence à l'adresse 2256. L'indication du caractère courant du buffer se trouve à l'adresse 2252. Ce pointeur est lu ligne 30 et est affecté à la variable X. Si on entre maintenant un autre caractère, ce pointeur est augmenté de 2. Cette modification est constatée ligne 50. Le nouveau pointeur est alors lu et additionné à l'adresse de début du buffer. A l'adresse pointée se trouve le nouveau caractère que nous obtenons par un PEEK(2256+X). Le caractère est ensuite affiché, et le programme continue son exécution. Cet exemple montre clairement que le programme n'attend pas qu'une touche soit enfoncée, mais tourne constamment.

Nous allons utiliser cette astuce dans notre programme pour créer un terminal supportant l'environnement duplex.

```
10  rem  ** terminal duplex **
20  p=2552
30  dim a%(5000)              : rem réserver de la place
40  i=0                       : rem compteur à zéro
50  z=peek(p)                 : rem lire le pointeur de pile
60  if inp(-1)=0 then 100     : rem recevoir un caractère ?
70  I=I+1                     : rem si oui, incrémenter le
                                compteur
80  a%(i)=inp(1)              : rem mémoriser le caractère
90  print chr$(a%(i));        : rem et afficher

100 if peek(p)=z then 60      : rem touche enfoncée ?
110 z=peek(p)                 : rem si oui, lire un nouveau
                                pointeur
120 x=peek(2256+z) and 255: rem et déterminer le
                                caractère
130 if x=13 then print        : rem éventuel saut à la ligne
140 print chr$(x);            : rem afficher le caractère
150 out 1,x                   : rem le transmettre
160 i=i+1 : a%(i)=x           : rem et le mémoriser
170 if x<>asc('*') then 60    : rem fin ?

180 for j=1 to i              : rem si oui, afficher le tout
190 print chr$(a%(j));
200 next j
```

Dans les lignes 10 à 50, on effectue toutes les initialisations. A la ligne 60, on interroge l'interface série pour savoir si un caractère a été collecté. Si c'est le cas, ce caractère est mémorisé dans le vecteur A% et affiché à l'écran. Les lignes 100 à 160 testent le buffer du clavier et traite un éventuel caractère saisi. Ces caractères sont également mémorisés dans le vecteur A%, afin de pouvoir par la suite lire la totalité de la conversation. Mais si le centre serveur travaille en mode écho, les lignes 140 et 160 peuvent être supprimées, car les caractères saisis sont de toute façon renvoyés et de ce fait mémorisés et affichés.

Si l'on entre le caractère étoile, le programme est interrompu et tout le texte qui a été mémorisé est affiché. A cet endroit, on peut remplacer PRINT par LPRINT, si on veut diriger les données vers l'imprimante et garder une trace écrite de la 'discussion'.

7.7. REDEFINITION DE CARACTERES

Le jeu des caractères de l'ATARI ST englobe une multitude de caractères et de symboles. C'est ainsi qu'il comprend les lettres grecques, ce qui est très intéressant pour les physiciens. Les symboles mathématiques comme la racine carré, l'intégrale ou le 1/2 en font également partie. On peut donc directement afficher des textes scientifiques.

Cependant, pour certaines applications, il serait utile de redéfinir des caractères pour pouvoir utiliser des symboles particuliers. L'environnement graphique des jeux peut alors être réalisé par une simple suite de PRINT CHR$(), ce qui à la fois augmente la vitesse d'affichage et facilite la programmation.

Cependant, le BASIC n'offre pas de fonction permettant la redéfinition d'un caractère. On doit donc à nouveau avoir recours à PEEK et à POKE pour le réaliser directement.

Dans le système d'exploitation, 3 jeux de caractères sont implantés. Ils possèdent tous les même symboles mais une résolution différente. En ce qui concerne l'affichage sur l'écran du moniteur monochrome, un jeu de caractères 8*16 est prévu, c'est à dire que chaque caractère est inscrit dans un rectangle de 8 points de large sur 16 points de haut.

En couleur, la résolution n'est plus que de 8*8, la résolution verticale étant diminuée de moitié. Enfin, il y a encore le jeu 6*6. Ce dernier jeu est utilisé pour annoter les icônes qui indiquent le type des fichiers du répertoire.

Pour pouvoir maintenant manipuler un de ces jeux de caractères, il faut d'abord savoir où il se trouve en mémoire. Dans la version du TOS qui est chargée à partir de la disquette, le début du jeu 8*16 se trouve à l'adresse 101474. A partir de cette adresse les 256 premiers octets correspondent à la ligne supérieure des 256 caractères du jeu, puis suivent les données de la ligne suivante, etc... Le bit de plus fort poids du premier octet d'un caractère détermine le point supérieur gauche de sa représentation graphique. Dans le jeu normal, la ligne supérieure est toujours égale à 0, donc blanche. Cela empêche que deux caractères se trouvant l'un au dessus de l'autre ne se touchent, car cela rendrait le texte plutôt illisible.

Nous allons examiner un programme qui modifie le caractère affichable par PRINT CHR$(1). Ce caractère est utilisé dans le coin supérieur droit d'une fenêtre graphique et représente le symbole d'ouverture d'une fenêtre. On obtient également ce caractère en entrant au clavier Control-A.

Le programme utilise la conversion binaire-décimal que nous avons déjà étudiée. La différence avec le programme redéfinissant le motif de remplissage au chapitre 7.1.5 réside dans le fait que seuls 8 bits sont ici manipulés. Sinon, le principe est le même. Le caractère créé représente à nouveau un bonhomme, mais on peut naturellement choisir un autre motif.

Attention : Ce programme ne fonctionne qu'avec le TOS en RAM et non avec le TOS en ROM.

```
10  rem ** redéfinition d'un caractère **
20  def seg=1                    : rem on manie des octets
80  for i=0 to 12 : read x$      : rem ligne du motif
82  x=0 : for j=1 to 8           : rem conversion
84  x=x-(mid$(x$,j,1)<>' ')* 2^(8-j)
86  next j
90  poke 101474+i*256,x          : rem mise en place du motif
100 next i
```

```
110 print chr$(1)                : rem affichage d'essai
120 end 130 rem +++ données du motif +++
150 data '   ***    '
160 data '* * * *   '
170 data '*  ***    '
180 data '**  *     '
190 data ' *******  '
200 data '  ******  '
210 data '   *** *  '
220 data '  ******  '
230 data '  ** **   '
240 data '  ** **   '
250 data '  ** **   '
260 data '  ** **   '
270 data ' *** ***  '
```

Si on veut modifier un autre caractère, il suffit de changer la valeur 101474 qui se trouve ligne 90. Pour la réalisation d'un nouvel A, c'est à dire CHR$(65), il suffit d'additionner 64, car 101474 est l'adresse du caractère numéro 1. Vous pouvez donc de cette manière créer votre propre jeu de caractères. Je vous conseille de le sauvegarder sur disquette par un BSAVE 'MONJEU.FNT',101474,4096 par exemple, pour pouvoir le recharger ultérieurement avec BLOAD 'MONJEU.FNT',101474.

Le nom de fichier est choisi dans cet exemple de manière arbitraire. Vous pouvez sauvegarder autant de jeux de caractères que vous voulez dans des fichiers différents, afin d'utiliser pour chaque application un jeu particulier. On peut ainsi créer des jeux vidéos ou bien écrire des textes en alphabet cyrillique. Les possibilités d'application n'ont donc presque pas de limites.

Encore un conseil :

Avant de créer ou de charger un jeu de caractère particulier, vous devriez d'abord sauvegarder le jeu standard. Si vous n'avez plus besoin des caractères redéfinis, vous êtes sûr, en chargeant le jeu de caractères standard, que vous pourrez à nouveau lire le répertoire de la disquette ou le menu du desktop...

Il y a un autre moyen pour modifier une police de caractères. Pour cela, il faut copier les données de la police de caractères dans la mémoire RAM et changer un pointeur du système d'exploitation. Il s'agit du pointeur que le TOS utilise pour avoir accès aux données de la police de caractères.

Cette méthode a deux grands avantages :

1) Si la police de caractères originale se trouve en ROM, elle ne peut pas être modifiée. Mais les données en RAM peuvent être chargées, sauvegardées et modifiées à volonté.

2) Le pointeur qui est utilisé dans l'exemple suivant ne convient que pour l'affichage avec la fonction OUT. Ainsi, on dispose de deux types de polices, l'une étant accessible avec PRINT et l'autre avec OUT 2,x: On peut donc mélanger les deux polices.

Voyons un exemple d'application de la méthode décrite :

```
10 defbl a,b,c                  :'préparer le pointeur
20 a=&H19858                    :'Adresse de la polixce d'origine
30 c=&H2924                     :'Adresse du pointeur
40 x=a                          :'copier le pointeur de police
50 dim a%(2050)                 :'réserver de la place
60 for i=0 to 2050
70 a%(i)=peek(x+i*2)            :'copier les données de la police
80 next
90 end

100 c$="A"                      :'caractère à modifier
110 n=&H66                      :'Bits de la ligne supérieure
120 b=varptr(a%(0))             :'Adresse de la nouvelle police
130 poke c,b                    :'dériver le pointeur
140 m=&HFF                      :'Préparer le masque
150 o=asc(c$)
160 if (o mod 2)=1 then m=m*256 else n=n*256
170 a%(o/2)=a%(o/2) and m or n

180 gosub 300                   :'modifier le texte d'essai
190 poke c,a                    :'installer la police d'origine
200 gosub 300                   :'texte d'essai original
210 end
```

```
300 restore
310 for i=1 to 9                    :'afficher texte d'essai "ABC"
320 read x
330 out 2,x: next
340 return
350 data 27, 72, 27, 66, 27, 66, 65, 66, 67
```

La première partie de ce programme copie la police d'origine dans le vecteur d'entiers a% après avoir préparé le pointeur utilisé. Il faut remarquer que chaque valeur de ce vecteur contient deux octets de la police de caractères.

Pour cette raison, il faut utiliser un masque afin de ne pas modifier l'octet voisin. Ce masque est mis dans la variable m dans la seconde partie du programme. Avec ce masque, l'octet à modifier est isolé et on lui donne la valeur n. Dans l'exemple, on met la valeur $66, ce qui transforme le A en A.

La troisième partie affiche un texte d'essai "ÄBC" qui est écrit "ABC". Lors du retour à la police d'origine, le texte d'essai est affiché de nouveau et est alors écrit "ABC".

Avec cet exemple, on voit comment utiliser simplement une nouvelle police de caractères. Seule la création des nouvelles données de la police est un peu compliquée.

Mais le programme qui suit pourra vous aider. Il s'agit d'un éditeur de caractères très simple qui, bien que pas particulièrement rapide (BASIC !), est très facile à utiliser grâce à la souris.

Lors de la mise en route du programme, un rectangle apparaît à l'écran, composé de 8 fois 16 points ou étoiles. D'autre part, le caractère courant est affiché sous la forme habituelle. Ainsi, on peut modifier directement le nouveau caractère.

Une étoile correspond à un point allumé (noir) dans la représentation du caractère. On appelle un tel point isolé un pixel et un caractère est composé de 8*16 soit 128 pixels.

Outre ce motif, le caractère d'origine est affiché sans modification avec l'instruction PRINT.

D'autre part, on voit sous le caractère d'origine les symboles "<" et ">". Si on clique sur un de ces symboles, le caractère précédent ou suivant est affiché.

La modification d'un caractère est donc très simple. Si on clique sur une étoile (point allumé), elle est remplacée par un point. En même temps, ce pixel disparaît dans le motif du caractère. De même, un pixel est de nouveau allumé si on clique sur le point correspondant.

```
10 '** EDITEUR DE CARACTERES **
20 dim a%(2050)
30 defdbl a,b,c
40 a=&H19858: c=&H2924
50 d=a
60 for i=0 to 2050
70 a%(i)=peek(d+i*2)              :' copier la police
80 next
90 b=varptr(a%(0))                :' pointeur sur nouvelles données
100 fullw 2: clearw 2
110 poke c,b                      :' installer nouvelle police
120 ch=65                         :' premier caractère "A"

130 poke contrl,123: vdisys 0     :' effacer le pointeur de souris
140 gotoxy 12,2
150 ?" caractère :",chr$(ch)      :' caractère d'origine
160 gotoxy 12,4: ?"< >"
170 if (ch mod 2)=0 then o=8 else o=0
180 gotoxy 0,0: ?"_____"
190 for i=0 to 15                 :' dessiner le masque
200 gotoxy 0,i+1: ?"|";
210 x=a%(ch/2+128*i)
220 for j=0 to 7
230 if (x and 2^(7-j+o))=0 then ?"."; else ?"*";
240 next j: ?"|"
250 next i
260 ?" ———"
```

```
270 out 2,27: out 2,asc("Y")
275 out 2,36: out 2,45          :' Positionner le curseur
280 for i=1 to 5: out 2,ch      :' 5 nouveau caractère
285 next
290 poke contrl,122: vdisys 0   :' allumer le pointeur de souris

300 mx=int(peek(9952)/8)-1
310 my=int((peek(9954))/16)-4 :' obtenir la position de la souris
320 if (peek(3581) and 1)=1 then 420 :'touche droite souris
330 if (peek(3581) and 2)=0 then 290  :'touche gauche souris
340 if my=3 and mx=24 then ch=ch-1: goto 130
350 if my=3 and mx=27 then ch=ch+1: goto 130
360 if mx>8 then 300

370 x=a%(ch/2+128*my)
380 m=2^(7-mx+o)                :'bit cliqué
390 if (x and m)=0 then x=x or m else x=x and (m xor &HFFFF)
400 a%(ch/2+128*my)=x           :'renouveler
410 goto 130

420 gotoxy 0,18
430 input "nom du fichier :",f$          :'sauvegarder police
440 bsave f$,b,4100
```

Dans les lignes 420 à 440 la nouvelle police de caractères est sauvegardée sur disquette. Ainsi, elle peut être mémorisée si besoin est et activée en repositionnant le pointeur $2924. Avec cette méthode, rien ne limite l'utilisation de polices de caractères "exotiques" pour vos programmes !

ANNEXE 1.

GLOSSAIRE

Adresse

Toutes les cellules mémoires adressables de la mémoire de travail sont numérotées dans l'ordre. Ce numéro permet d'accéder (lire/écrire) au contenu de la cellule mémoire ; il représente l'adresse de celle-ci.

Application (programme d')

Un programme qui est directement exécutable est appelé programme d'application.

ASCII (American Standard Code of Information Interchange)

Comme un ordinateur ne peut traiter que des valeurs binaires, il faut faire correspondre à chaque caractère alphanumérique une valeur binaire. Le code le plus utilisé est le code ASCII sur 7 resp. 8 bits qui permet de manipuler 128 resp. 256 caractères.

Assembleur

On désigne par assembleur le programme qui génère, à partir du programme machine écrit sous forme de mnémoniques, le code objet du processeur correspondant.

Bibliothèque

Une bibliothèque est un ensemble de sous-programmes, de fonctions et de routines utilitaires qui peuvent être linkées à un programme.

Bit (Binary Digit)

Un ordinateur ne peut traiter que des informations binaires, c'est-à-dire des informations pouvant uniquement prendre les deux valeurs 0 ou 1, que l'on désigne aussi par LO et HI. Les nombres binaires (bits) sont les plus petites unités d'information dans la technique des ordinateurs.

Buffer (zone tampon)

Un buffer est une zone mémoire qui contient des données de manière provisoire. Il permet par exemple de combiner deux processus de vitesse différente : le processus le plus rapide y stocke des informations que l'autre peut lire plus tard et plus lentement.

Bus

Un bus est un ensemble de fils permettant l'interconnexion entre les différents organes d'un ordinateur. Exemple : la CPU et la mémoire de travail sont reliées par le bus de donnée, le bus d'adresse et le bus de contrôle ; les bus de périphérique relient l'ordinateur aux différents périphériques comme par exemple l'imprimante. Par ailleurs, on distingue les bus unidirectionnels (la transmission des données n'a lieu que dans un sens) et les bus bidirectionnels (transmission dans les deux sens).

C (langage de programmation)

C est un langage de programmation évolué qui possède cependant des caractéristiques proches du langage machine. Il est très rapide, proche de la machine et, en comparaison avec le langage machine, plus simple à apprendre. La structure du langage ressemble à celle d'ALGOL ou de PASCAL. Les programmes écrits en C sont facilement transportables sur d'autres ordinateurs, si ces derniers possèdent également un compilateur C. Cette propriété a, pour la première fois, permis d'implanter un même système d'exploitation (comme UNIX ou GEM) sur des ordinateurs ayant des CPU différentes, sans avoir à le réécrire.

Caractère de contrôle

Pour prendre en compte les fonctions spéciales de l'imprimante et de l'écran, on utilise soit les codes ASCII non imprimables (0 à 31), soit les séquences appelées séquences Escape (La touche Escape suivie de n'importe quelle autre touche).

Cartridge

Les cartridges sont des cartouches avec des ROM allant jusqu'à 128 Ko, qui peuvent être insérées dans la fente sur le côté gauche de l'ATARI ST. Elles contiennent des programmes d'application ou des extension du système d'exploitation.

Centronics (interface)

Cette interface a été introduite par la société Centronics pour la connexion d'imprimantes et est devenue une interface standard. Il s'agit d'une interface unidirectionnelle, transmettant 8 bits en parallèle, et avec une liaison Handshake. Sur l'ATARI ST, cette interface est bidirectionnelle.

Champ de contrôle

Fenêtre de dialogue, dans laquelle différents paramètres peuvent être définis, comme par exemple les couleurs, l'heure et le beep du clavier.

CPU (Central Processing Unit)

Dans les micro-ordinateurs, on désigne par CPU le microprocesseur. L'ATARI ST utilise le MC 68000 de Motorola.

Curseur

On désigne par curseur différentes marques (flèche, abeille, réticule ou bloc) qui sont déplacées sur l'écran grâce à la souris (curseur de la souris) ou au clavier (curseur texte), et qui déterminent la prochaine position de saisie.

Debbuger

Il s'agit d'un programme utilitaire pour la recherche et la correction des erreurs dans un programme machine assemblé ou compilé. Un debugger permet entre autre d'interrompre l'exécution du programme (Break-Points), et de lire ou modifier le contenu de la mémoire. (SID)

Débit

Le débit désigne généralement la vitesse de transmission des données pour l'interface série RS 232. Un débit de 300 Baud, par exemple, signifie une vitesse de transmission de 300 bit/s (bit par seconde). Avec l'ATARI ST, on peut définir des débits d'information entre 300 et 9600 baud.

Désassembleur

Utilitaire qui permet de traduire le code-machine en mnémoniques. Il facilite la correction des erreurs et la modification des programmes dont on n'a pas de listing.

Desktop (bureau)

L'écran principal sous GEM, qui contient le menu principal, les symboles pour les disquettes et la corbeille à papier.

Dialogbox (fenêtre de dialogue)

On désigne par fenêtre de dialogue une fenêtre interactive qui affiche certaines informations et attend une réponse.

Disque dur (Hard-Disk)

Un disque dur est un moyen de stockage qui utilise le même procédé magnétique que les disquettes. En comparaison avec ces dernières, la capacité d'une unité de disque dur est multipliée par un facteur 10 à 100 (5 à 500 Moctets), et le débit de transmission par un facteur 10. Cela est dû principalement au fait que la 'disquette' est installée de façon stable et définitive à l'intérieur du lecteur et travaille ainsi avec une vitesse de rotation bien supérieure (environ 10 fois).

DMA (Direct Memory Access)

DMA veut dire 'accès direct à la mémoire' et désigne la capacité des composants périphériques à lire/écrire dans la mémoire, sans l'intervention du CPU.

Duplex

Duplex désigne une procédure de transmission de données, par laquelle il est possible de transmettre des informations, soit dans les deux sens (Full Duplex), soit en changeant presque instantanément le sens de transmission.

Emulation

On entend par émulation le processus permettant de simuler un autre appareil au moyen de software et/ou de hardware supplémentaire. Ainsi, sur l'ATARI ST, on peut par exemple simuler un terminal-VT52.

Entrée/Sortie voir I/O

Fichier (File)

Un fichier est un ensemble de données stocké sur disquette ou sur disque dur. L'accès à ces données se fait grâce à un nom spécifique.

Folder (sous-directory)

Dans le répertoire de la disquette, on peut définir avec Folder des sous-directories (sous-répertoires) spécifiques et indépendantes. Celles-ci peuvent contenir des fichiers qui ne sont plus gérés par le répertoire principal. L'accès à ces fichiers est possible après ouverture des sous-directories correspondantes.

Formattage

Avant de pouvoir stocker des informations sur un support magnétique (disquette ou disque dur), il faut le formatter, c'est-à-dire le diviser en pistes et secteurs.

GDOS (Graphic Device Operating System)

Le GDOS contient les fonctions graphiques du GEM-VDI, qui sont indépendantes de l'ordinateur utilisé.

GIOS (Graphic Input/Output System)

Cette partie du GEM-VDI contient le code spécifique à l'ordinateur.

Hardcopy

Sortie sur imprimante du contenu de la mémoire écran momentanée.

Hexadécimal

Le système hexadécimal permet de représenter des nombres en base 16. C'est, à côté des représentations binaire et octal, le système numérique le plus utilisé en informatique.

I/O (Input/Output = Entrées/Sorties)

Cette notion désigne les informations qui proviennent des périphériques (mémoire de masse, clavier, souris, etc...) ou qui sont émises vers ceux-ci (écran, imprimante, table traçante, etc...).

Interface

Une interface est d'une part la liaison électronique qui permet de connecter un périphérique à l'ordinateur, et d'autre part le programme qui permet de faire le lien entre différents programmes indépendants, ou entre l'utilisateur et un programme (par ex. le GEM entre l'homme et le TOS).

Interruption

Une interruption est, comme son nom l'indique, un arrêt provisoire de l'exécution du programme en cours. Elle branche le compteur ordinal sur une routine machine. Cette dernière une fois terminée, on rend la main au programme interrompu, à l'endroit où a eu lieu l'interruption (comme pour un sous-programme). Cette interruption est déclenchée par un évènement 'hard' à un moment précis ou, de manière 'soft' par un programme (par ex. l'instruction TRAP).

K (Kilo-octet)

Un kilo-octet devrait être 1000 octets, mais en informatique, on évalue le plus souvent en puissance de 2 ; de ce fait, un Koctet désigne en réalité 2^10=1024 octets.

Linker (éditeur de liens)

Le linker est un programme utilitaire qui permet de relier des programmes compilés, des routines machines et des éléments de la bibliothèques à un programme exécutable. Cette opération a lieu après la compilation.

Menu (liste d'options)

Un menu propose dans un programme plusieurs options possibles. L'une d'entre elles peut être choisie en entrant le caractère correspondant ou la validant avec le curseur de la souris.

MIDI (interface MIDI)

L'interface MIDI est une interface normalisée, série qui permet de piloter à partir de l'ordinateur des instruments de musique. Plusieurs instruments de musique peuvent être connectés simultanément.

Mnémonique

Le véritable langage machine est uniquement composé de 0 et de 1. Les instructions étant elles-mêmes constituées de ces deux chiffres, elles sont difficilement lisibles et manipulables. Pour cette raison, on a introduit une suite de 3 ou 4 lettres pour désigner une instruction. Ainsi par exemple, la séquence de 0 et de 1 définissant l'instruction machine d'addition a été remplacée par la mnémonique ADD. Les programmes écrits en mnémocode doivent être traduits (ce programme traducteur est appelé assembleur) en un programme exécutable.

Mot

Pour l'ATARI ST, un mot représente une cellule mémoire qui a la taille de 2 octets, donc 16 bits. Un mot peut donc prendre une valeur comprise entre 0 et 65535.

Objet (code objet)

Le code objet représente le code programme directement exécutable par la CPU. Il est uniquement composé de bits ou si, l'on veut, de nombre hexadécimaux. On obtient ce code par la compilation d'un programme écrit en langage évolué ou par l'assemblage du mnémocode.

Octet

Un octet est constitué de 8 bits. Il représente, dans la plupart des micro-ordinateurs et des mini-ordinateurs, la plus petite cellule mémoire adressable, même si la CPU et la mémoire, comme par exemple dans l'ATARI ST, sont organisées en mots de deux octets (16 bits) et en mots longs (32 bits).

Output

L'output est l'émission de données vers les périphériques.

Parallèle (Interface parallèle)

Voir Centronics

Paramètre

Les paramètres sont des variables ou des constantes qui sont transmises à des instructions, fonctions ou sous-programmes. Plusieurs paramètres peuvent être nécessaires pour une seule fonction, par ex. LINEF A,B,C,D.

PEEK

L'instruction-BASIC PEEK lit le contenu d'une cellule mémoire directement adressée et le transmet au programme appelant. Cette instruction peut traiter des octets, des mots et des mots longs.

Périphérique

Les appareils externes d'un système informatique, comme l'imprimante, le lecteur de disquettes ou l'écran, sont appelés périphériques.

Pixel

Le pixel est le plus petit élément graphique que l'on peut adresser sur l'écran (point d'écran) ou sur l'imprimante (point de matrice).

POKE

Cette instruction BASIC représente l'inverse de l'instruction PEEK. Elle permet de modifier directement une cellule mémoire. Ici également, on peut adresser des octets, des mots et des mots longs. Un seul mauvais POKE peut 'planter' tout le système, si il accède au système d'exploitation.

RAM (Random Access Memory)

RAM signifie littéralement 'mémoire à accès quelconque'. Cette mémoire peut être lue et modifiée. Usuellement, la notion de RAM indique la mémoire faite de semi-conducteurs.

Registre

Un registre est une cellule mémoire interne au microprocesseur, dans lequel des données peuvent non seulement être mémorisées mais encore être traitées (addition,etc...). Le MC 68000 possède en tout 16 registres de 32 bits qui sont tous utilisables.

RGB (Red-Green-Blue)

Il s'agit d'un signal vidéo qui permet d'émettre les trois signaux de couleur au téléviseur ou moniteur, que ce dernier mixe de manière additive. Avec l'ATARI ST, chaque couleur de base peut prendre 8 valeurs d'intensité différentes, de telle sorte que 8*8*8=512 couleurs peuvent être représentées.

ROM (Read Only Memory)

La ROM est une partie de la mémoire, qui peut uniquement être lue, à l'inverse de la RAM. La programmation des composants constituant la ROM est réalisée une fois pour toute lors de sa production. La ROM contient souvent le système d'exploitation ; ce dernier peut ainsi être activé dès la mise sous tension de l'appareil.

RS 232

Cette interface travaille en série et est normalisée. La transmission des données est possible dans les deux sens. Les tensions utilisées sont +12V et -12V.

Scrolling

On désigne par scrolling le déplacement du contenu de la fenêtre écran dans les quatre directions. Dans l'ATARI ST, ce déplacement est géré par le système d'exploitation quand l'utilisateur choisit de déplacer une fenêtre avec la souris.

Sous-directory

(voir Folder)

TOS (Tramiel Operating System)

Le système d'exploitation utilisé dans l'ATARI ST, le TOS, est une extension du CP/M 68K ; des fonctions supplémentaires y ont été ajoutées. Cependant, le TOS n'est compatible avec aucun autre système d'exploitation.

VDI (Virtual Device Interface)

Le VDI est une partie du GEM qui gère les sorties graphiques vers un périphérique quelconque.

VT52-Terminal

Cette option du menu du desktop permet de simuler un terminal qui peut être directement connecter, via l'interface série, à un modem ou à un autre ordinateur.

ANNEXE 2.

PEEK ET POKE UTILES

La liste des adresses PEEK et POKE qui suit n'est pas seulement un résumé. Elle sert également à l'adaptation des programmes au système d'exploitation utilisé.

La première valeur correspond au système d'exploitation en date du 11/1985 et qui fait environ 197 Koctets sur disquette. Nous avons utilisé ces valeurs dans le livre.

La deuxième valeur correspond à l'adresse servant pour le premier TOS disponible qui était lui aussi chargé sur disquette.

La troisième adresse est valable pour le TOS sur ROM, c'est à dire pour la version d'avril 1986.

ADRESSE(S)			FONCTION
3086,	2256,	3086	Tampon clavier
3510,	2552,	3510	Pointeur sur tampon clavier
3581,	2623,	3581	Statut des touches souris/joystick
3582,	2624,	3582	Position du joystick
3592,	2634,	3592	Etat du joystick
9952,	8994,	9952	Position horizontale de la souris
9954,	8996,	9954	Position verticale de la souris
10530,	9572,	10530	Vitesse de clignotement du curseur
10531,	9573,	10531	Compteur de clignotement du curseur
10532,	9574,	10532	Pointeur sur les données de style
36612,	32090,	16527388	Données sonores pour la cloche
36642,	32120,	16527418	Données sonores pour le clic
104536,	101474,	16595294	Début du caractère 8*16

ANNEXE 3.

TABLE DES MOTS CLES

ACIA	1.1
AES (programmation de l')	7.3
AES	4.2
Accès privilégié	6.4.1
Adressage (mode d')	6.4
Alarme (fenêtre)	7.3
Array	4.2.1
BDC (code)	3.3
BDC (conversion)	1.3.4
BIOS	4.1
Beep du clavier	7.2
Binaire (arithmétique)	3.3
Branch	6.4
Bruit	7.2
Bruit de fond	7.2
Bubble-Sort	7.4
C	6.3
CALL	6.4.1
Caractère (type de)	4.2.4
Cartbridge	1.2.5
Centre serveur	7.6.3
Centronics	1.2.1
Cercles	7.1.1
Chargement/sauvegarde de données	2.2.1
Chemin	7.6.2
Clavier (buffer du)	7.6.3
Clavier (programmation)	1.3
Clavier	1.3
Cloche (bruit de)	7.2
Compilateur	4.3
Configuration de la mémoire	2.1
Connexion d'un autre lecteur de disquettes	1.2.3
Contrôleur	1.2.3
Conversion entre systèmes numériques	3.1
Couleurs (mélange de)	7.1.8

Curseur (clignotement) 4.1.3
Curseur (déplacement avec la souris) 1.3.3

DEF SEG 7
DMA 1.1
Dessin avec POKE 2.1.4
Dessin de marques 7.1.3
Dessin en LOGO 1.4.1
Diagramme rectangle 7.1.1
Direction d'écriture 7.1.2
Disquette (accès) 7.6.2
Disquette 2.2
Données (type de) 6.3
Duplex 7.6.3
Débit de transmission (configuration) 5.1
Dérivation du programme 4.1.3

Ecriture (taille de l') 7.1.2
Effacement de l'écran 6.4
Enveloppe (courbe) 7.2
Escape (séquence) 4.1.3
Extrémités des droites 7.1.3

Fichier (choix de) 7.6.2
Fichier (gestion) 7.4
Fichier (type) 7.6.2
Fichiers sur disquette chargés dans des chaînes de caractères 7.4
Formatage 1.2.3

GEM (déconnexion) 4.2.5
GEM 4.2
GEMDOS 4
GLUE 1.1
Graphique-texte 4.2.5
Graphisme 7.1

Heure/date 1.3.4
Hexa (Dump hexa) 7.6.2
Hexadécimal 3.1

I/O (domaine) 2.1
INKEY$ 7.6.3
Imprimante (paramètres) 7.6.1
Imprimante (programmation) 7.6.1
Imprimante (simulation) 1.2.1
Interface 1.2
Interprèteur 4.3
Interruption 7.2

Jeu de caractères 7.7
Jeu de caractères 7.7
Joystick (lecture de l'état) 3.2
Joystick (pilotage) 7.5
Justification de textes 4.2.5

LPRINT en LOGO 6.1
Lignes (types de) 7.1.3
Lignes (épaisseur des) 7.1.3

MIDI 1.2.4
MOVE 6.4
Machine (langage) 6.4
Marquage 7.1.6
Masque 7.4.1
Menu (réalisation) 7.3
Mnémonique 6.4
Modem 7.6.3
Moniteur 1.2.7
Motif de remplissage 7.1.4
Mémoire 2.1
Mémoire écran 2.1.4

Opérateurs logiques 3.2

PAL/SECAM (mise en place) 1.2.7
PEEK et POKE 7
Pile (pointeur de) 6.4
Pile 2.1.5
Plantage 2.1.3
Pointeur 2.1.4
Processeur 1.1

Programme machine affecté à une variable	6.4.1
Protection	6.4
RAM/ROM	2.1
RGB	1.2.7
RS 232	1.2.2
Réseau d'ordinateurs	1.2.4
SCART	1.2.7
SWAP	7.4
SYSTAB	4.2.5
Sauvegarde de l'image	2.2
Secteur (lecture)	7.6.2
Secteur/Track	2.2
Shifter	1.1
Son (séquence sonore)	7.2
Sound	7.2
Sound-chip (programmation)	7.2
Souris (changement de forme)	4.2.3
Souris (forme du curseur)	4.2.3
Souris (positionner le curseur).	7.5
Souris (état de la)	7.3
Souris	1.4
Souris (interrogation de la position)	4.2.2
Superviseur	6.4
Surface (remplissage)	7.1.4
Synchronisation	1.2.7
Système d'exploitation	4
Systèmes numériques	3.1
TOS (curseur)	4.1.3
TOS	4.1
TOS-ERROR (fenêtre)	7.3
TRAP (vecteur)	2.1.3
TRAP	6.4
Terminal (programme)	7.6.3
Test de la couleur d'un pixel	7.1.7
Touches-fonctions	1.3.5
Traitement de textes	7.4
Tri	7.4
Turtle (graphique)	6.1
Téléviseur (branchement)	1.2.7

Utilisateur (mode) 6.4

VARPTR 2.1.4
VDI (programmation) 4.2.1
VDI 4.2
VT52 (émulateur) 4.1.3
Variable système 4.1.2
Vecteur 2.1.4
Vecteurs exceptions (pointeurs erreurs) 2.1.3

WAVE 7.2

XBIOS 4.1

les livres Atari ST

LE LIVRE DU GEM SUR ATARI ST

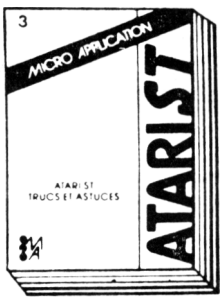
Cet ouvrage contient ce qu'il est necessaire de savoir pour utiliser GEM efficacement fonctionnement de la souris, Virtual Device Interface, Application Environment Services, Graphics Device Operating System Description de routines utilisant GEM en C et en Assembleur Utilisation standard et specifique de l'operating system
Ref ML139
Prix **149** FF

LE LIVRE DU LANGAGE MACHINE DE L'ATARI ST

Tout ce qu'il faut savoir pour tirer au mieux parti de votre ATARI ST système de calcul et de bit manipulation du 68000, utilisation des registres, structure des commandes, programmation structurée récursion, piles, procédures et fonctions listings sources de programmes assembleurs, routines systèmes Un super livre!
Réf ML141
Prix **149 FF**

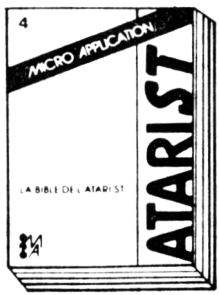
ATARI ST TRUCS et ASTUCES

Un recueil complet de trucs et d'astuces que votre tout nouveau ATARI ST va beaucoup apprecier! Des graphismes fantastiques a partir de programmes en BASIC, des exemples et des conseils pour programmer en langage C et en Assembleur...
Ref ML140
Prix **149** FF



LA BIBLE DE L'ATARI ST

Ce livre contient un ensemble complet d'informations sur l'ATARI ST, la description HARDWARE de la machine ainsi que des schemas detailles et amplement expliques, notamment des interfaces V24, du port d'extension, de l'interface midi, la structure des graphiques, du BIOS, de GEM, les adresses systemes importantes, le fonctionnement de la souris
Ref ML142
Prix **249** FF



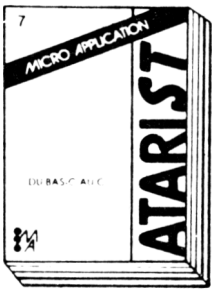
PEEKS & POKES

Voulez-vous profiter rapidement de votre ATARI ST? Bien sur, mais comment faire? Eh bien lisez cet ouvrage et vous decouvrirez vite tout ce que vous pouvez tirer facilement de votre ST Les domaines couverts sont la configuration hardware, la memoire, le TOS, GEM, les differents langages (LOGO, BASIC, C, Assembleur), programmation des graphismes et des sons, techniques de programmation, gestion des disquettes De nombreux exemples illustrent chaque chapitre
Ref ML153
Prix **129** FF

LE LIVRE DU BASIC

Ce livre est une introduction complete et didactique au BASIC de l'ATARI ST Tous les stades de la programmation sont couverts comme l'analyse du probleme, la realisation d'un ordinogramme, les differentes techniques de programmation Le lecteur abordera ensuite des domaines plus pointus mais passionnants comme la programmation des graphismes, la gestion des tableaux a plusieurs dimensions, les techniques de tri, la gestion des donnees, le BASIC
Ref ML154
Prix **149** FF

les plus de Micro Application



DU BASIC AU C

Vous voulez vous mettre au C ? Pas de problèmes ! Ce livre s'adresse à tous ceux qui ont une bonne connaissance du BASIC et qui veulent se mettre rapidement au langage C. Cet ouvrage leur permettra de transférer leurs connaissances acquises pour le BASIC au C et ainsi de rentabiliser leur investissement ! En une journée ils pourront déjà acquérir les fondements de la programmation dans ce super langage qu'est le C.

Ref : ML 155
Prix : **149** FF

BIEN DEBUTER AVEC L'ATARI ST

Ce livre s'adresse à tous ceux qui démarrent avec le ST et qui veulent réussir à coup sûr leurs débuts ! Ils pourront grâce à cet ouvrage éviter les premiers pièges et profiter immédiatement de leur nouvelle machine. Domaines couverts : installation du matériel, utilisation du TOS et de GEM, la souris, l'écran, le clavier, premiers programmes en BASIC, le langage LOGO, les différentes possibilités d'utilisation du S, etc.

Ref : ML 156
Prix : **129** FF
Disponible en Mai

GRAPHISMES & SONS

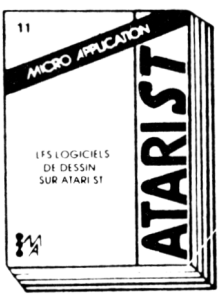
Avec ce livre vous allez pouvoir profiter pleinement des immenses capacités graphiques et sonores de l'ATARI ST. Ce livre est accompagné de nombreux programmes exemples et utilitaires écrits en BASIC, LOGO, C et MODULA 2. Quelques-uns des thèmes abordés : graphismes en 2 et 3 D, graphismes sous GEM, les rotations, les figures moirées, trucs et astuces du graphisme...

Ref : ML 157
Prix : **149** FF

LE LIVRE DU LOGO

Le LOGO est un langage très intéressant dont les applications sont très nombreuses. Cet ouvrage permettra au lecteur de profiter au maximum du LOGO livré avec l'ATARI ST. Principaux thèmes abordés : les graphismes, les procédures, les récursions, les routines de tri, un générateur de masque, structure des données, intelligence artificielle...

Ref : ML 158
Prix : **149** FF
Disponible en Mai



LES LOGICIELS DE DESSIN SUR ATARI ST

L'ATARI ST a de fantastiques capacités graphiques et plusieurs logiciels du commerce permettent de les exploiter et de réaliser de très beaux dessins. Pourtant ces programmes ne sont pas toujours très faciles à utiliser. Cet ouvrage vous dira tout sur GEM-DRAW, DEGAS et NEO-CHROME et vous livrera beaucoup de trucs et astuces qui vous faciliteront la tâche. Vous pourrez ainsi tirer partie au maximum de votre logiciel.

Ref : ML 159
Prix : **129** FF
Disponible en Juin

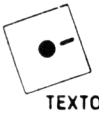
Disponible en Juin

LES GRAPHISMES EN 3 DIMENSIONS

Avec ce livre vous allez acquérir une connaissance complète de la programmation des graphismes en 3 D et de leur animation sur ATARI ST. Cet ouvrage vous enseignera tout d'abord les bases théoriques nécessaires à l'animation en 3D, puis vous expliquera clairement les notions de programmation en ASSEMBLEUR indispensables pour obtenir des résultats spectaculaires. Enfin vous passerez à la mise en pratique avec l'analyse et la réalisation de routines et programmes graphiques complets.

Prix : **179** Francs.
Réf. : ML 167

les logiciels Atari ST



TEXTOMAT ST

TEXTOMAT ST est le traitement de texte que tous les possesseurs d'ATARI ST attendaient avec impatience. Puissant, rapide, utilisant au maximum les fantastiques capacités des ST, TEXTOMAT ST peut reprendre à son compte le désormais célèbre slogan d'ATARI : "PLUS DE PUISSANCE POUR MOINS D'ARGENT".

Principales caractéristiques :

Très haut niveau de performances du à une saisie et un traitement des textes en mémoire ultra rapide.
Toutes les fonctions sont utilisables à partir de la souris ou du clavier (très pratique).
Jusqu'à 180 caractères par ligne avec scrolling horizontal automatique.
30 touches de fonctions définissables avec capacité de 160 caractères (formules de politesses, répétition de phrases, adresses...)

Tabulations illimitées.

Chargement et Sauvegarde des textes par RS232 (échanges de textes entre ordinateurs).
Editions possibles en colonnes avec numérotation automatique.
Impression verticale en A4 sur EPSON FX et compatibles.
Compatibilité avec TEXT DESIGN pour création de documents.
Génération automatique de tables des matières et index triés.
Toutes les fonctions classiques du traitement de texte (recherche, pagination, mouvements de paragraphes ...).

TEXTOMAT ST : un super logiciel, à un super prix pour une super machine.

REF : ST002
PRIX : 450 FF

Disponible en MAI!

DATAMAT ST

DATAMAT ST est une Gestion de Fichiers de haute qualité, facile à utiliser et spécialement développée pour ATARI ST.

Principales caractéristiques :

Utilisation totalement intégrée à GEM (menus déroulants, fenêtres...)

Fonction HELP.

Choix de différentes polices de caractères à l'écran.

Traitement simultané de plusieurs fiches.

Possibilité de travailler avec un RAM disk sur 1040 ST (très efficace et rapide).

Fonctionne avec disque dur.

Longueur d'enregistrements = 64 K.

Création de son propre masque de saisie.

Fichiers de 64000 enregistrements.

Compatible avec TEXTOMAT ST pour mailing, lettres types, adresses...

Format d'édition preétabli :
étiquettes, listes et paramétrable.

Recherche et extraction d'enregistrement très rapide.

Nombreuses procédures de TRI.

ETC...

Manuel détaillé en français.

REF : ST003
PRIX : 450 Francs

Disponible en JUIN.

les plus de Micro Application

MODULA-2/ST

MODULA-2/ST est un système de développement complet pour l'ATARI ST. Il est composé :

- d'un programme Editeur
- d'un compilateur Modula-2
- d'un éditeur de liens (linker)
- de librairies pré-définies

MODULA-2 gère la compilation séparée, les types opaques, les co-routines et les routines en virgule flottante. De plus MODULA-2/ST est intégré à l'environnement graphique GEM. Les intéractions entre l'éditeur, le compilateur, l'éditeur de liens sont dues aux menus GEM, à la sélection avec la souris et aux manipulations d'icônes.
MODULA-2/ST est un langage de haut niveau suffisamment simple pour permettre un démarrage facile aux novices de la programmation, mais suffisamment puissant pour que les programmeurs confirmés puissent écrire des logiciels complexes.

MODULA-2/ST vous permet de créer des programmes utilisant :

- Plusieurs polices de caractères
- Plusieurs fenêtres
- Des menus déroulants
- Toutes les possibilités graphiques de l'ATARI ST
- Les co-routines

Caractéristiques techniques :

- Compilation séparée avec contrôle des types.
- Support complet des routines GEM.
- Execution directe des programmes à partir du bureau (desk).
- Création de codes compactés.
- Documentation en français.
- Programmes de démonstration (graphiques).

REF : ST001
PRIX : 1450 FF

TEXT DESIGN ST

TEXT DESIGN est le complément idéal d'un logiciel de traitement de texte (notamment TEXTOMAT ST) pour réaliser des documents intégrants texte et graphiques.

TEXT DESIGN en quelques mots :

Chargement des données à partir d'un texte (fichier ASCII ou TEXTOMAT ST).

Opérations de blocs (copier, déplacer, détruire).

Traitement des graphiques.

Mélange de graphique et de texte.

Superposition des graphismes et du texte.

Commandes graphiques pour tracé de lignes et de cadres.

Possibilité de modifier les espacements entre mots.

Chargement et sauvegarde des documents réalisés.

Impression sur imprimante EPSON et compatibles.

Manuel détaillé en français.

REF : ST004
PRIX : 450 Francs

Disponible en JUIN.

BAKÉLITE ST

Logiciel d'aide à la création, l'élaboration, et la réalisation de circuits imprimés.

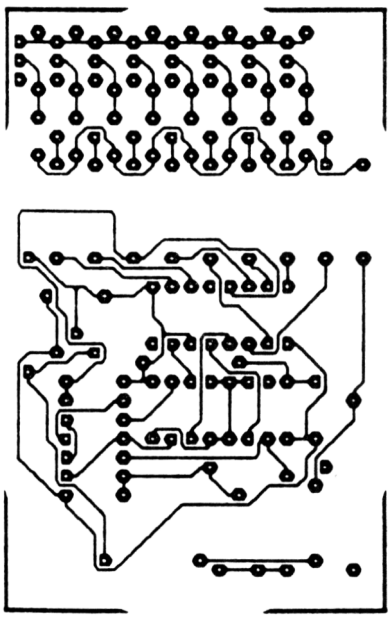
BAKELITE ST est un outil d'aide à la conception de circuit imprimé. Ce logiciel automatise tout le tracé de la carte du circuit imprimé en fonction de la mise en place des composants. Ce surprenant et remarquable logiciel réduit le coût, les longues heures fastidieuses et l'ennui qui accompagnent d'habitude la production d'une carte en état de marche et fonctionnant! Ce logiciel est une des premières applications verticales disponible pour l'ATARI ST et son degré de sophistication des commandes, ses qualités graphiques et sa remarquable rapidité en font un logiciel dont le prix devrait dépasser largement celui auquel Micro Application le propose.

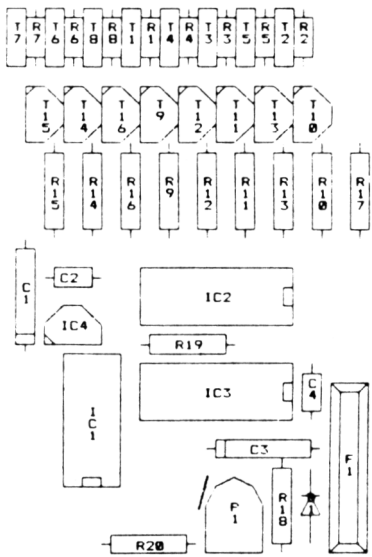
Ce qui est propre à tout les utilisateurs de BAKELITE ST c'est la conception automatique du tracé de tout le circuit et du cheminement entre chaque composant. Les connexions sont automatiquement dessinées à l'écran. Si le circuit est modifié, les connexions sont redessinées immédiatement. De plus, il est possible de sélectionner des tracés à 45 ou 90 degré, d'épaisseur plus ou moins importante, des connexions de broche à broche, de broche à BUS, de BUS à BUS, ou bien de traiter un circuit double face. BAKELITE ST imprime le schéma complet du circuit sur imprimante EPSON et compatibles. La haute précision d'impression permet la photogravure du document. De plus, BAKELITE ST imprime le schéma des composants placés sur le circuit ainsi que la liste de chaque composant et connexion.

Comment fonctionne BAKELITE ST ? Tout d'abord l'utilisateur définit les composants qui formeront le circuit. Par exemple, IC2 pourrait être un circuit intégré qui s'enfiche sur un support 14 broches à double rangées de connecteurs. Ce genre d'information est facilement saisie au clavier.
Puis il faut définir les connexions. Par exemple Broche 2 de IC2 doit être connectée au collecteur 1 du transistor 4. BAKELITE ST vous permet d'entrer, d'effacer ou de modifier ces informations. Les changements peuvent être effectués à n'importe quel moment donné durant la conception du schéma.

Lorsque ces différentes étapes ont été exécutées, BAKELITE ST vous demande de positionner les composants à l'écran. La souris du ST est utilisée pour placer chaque composant et bien sûr, chaque composant peut être déplacé à n'importe quel moment donné.

les plus de Micro Application





BAKELITE ST exécutera alors, automatiquement, à l'écran, le tracé des liaisons électriques entre chaque composant et connexion du schéma. BAKELITE ST imprimera ensuite le schéma complet sur une imprimante EPSON ou compatible et l'impression est alors prête à être filmée !

BAKELITE ST peut améliorer de façon conséquente la réalisation d'un schéma en éliminant beaucoup d'étapes superflues et en réduisant les temps de modification (qui ne sont pas les moindres !).

En tenant compte de toutes ces capacitées et des gains de temps réalisés, BAKELITE ST, sera rentabilisé dès la première conception d'un circuit imprimé.

BAKELITE ST fonctionne sur ATARI 1040 STF ou 520 ST équipé d'un moniteur monochrome.
Imprimante EPSON ou compatible sont recommandées.

REF : ST005 PRIX : 2450 FF Disponible en JUIN

Achevé d'imprimer en mai 1986
sur les presses de l'imprimerie Laballery
58500 Clamecy
Dépôt légal : mai 1986
Numéro d'imprimeur : 605030

IMPRIMÉ EN FRANCE

Voulez-vous profiter rapidement des possibilités offertes par le système d'exploitation de l'ATARI ST ? Eh bien lisez cet ouvrage et vous découvrirez vite tout ce que vous pouvez tirer facilement du ST ! Les différents domaines couverts sont notamment la configuration hardware, la mémoire, le TOS, GEM, les graphismes, la gestion des disquettes... De nombreux exemples commentent l'utilisation des POKES.

Aperçu du contenu du livre :

- **Coup d'œil sur le ST**
- **Configuration Mémoire et Interfaces**
- **Un clavier intelligent**
- **Une souris pour dessiner**
- **Pointeurs et Piles**
- **Gestion des disquettes**
- **Représentation interne des nombres**
- **Le TOS**
- **GEM et BASIC**
- **Interpréteur et Compilateur**
- **Les différents langages (C, LOGO, BASIC, Assembleur)**
- **La Programmation du ST en BASIC (graphismes, sons, fenêtres...)**
- **Programmer l'imprimante, le lecteur de disquette, les communications.**

ISBN : 2-86899-052-5

Prix : 129 FF
Réf. : ML 153